Advanced Denoising and Memoryless Acceleration for Realistic Image Synthesis

Von der Carl-Friedrich-Gauß Fakultät
der Technischen Universität Carolo-Wilhelmina zu Braunschweig

zur Erlangung des Grades eines

Doktoringenieurs (Dr.-Ing.)

genehmigte

Dissertation

von Pablo Bauszat

geboren am 02. März 1983

in Gießen

Eingereicht am: 18. Mai 2015

Disputation am: 07. August 2015

1. Referent: Prof. Dr.-Ing. Marcus Magnor

2. Referent: Prof. Dr.-Ing. Marc Stamminger

(2015)

Abstract

Stochastic ray tracing methods have become the industry's standard for today's realistic image synthesis thanks to their ability to achieve a supreme degree of realism by physically simulating various natural phenomena of light and cameras (e.g. global illumination, depth-of-field, or motion blur). Unfortunately, high computational cost for more complex scenes and image noise from insufficient simulations are major issues of these methods and, hence, acceleration and denoising are key components in stochastic ray tracing systems. In this thesis, we introduce two new filtering methods for advanced lighting and camera effects, as well as a novel approach for memoryless acceleration. In particular, we present an interactive filter for global illumination in the presence of depth-of-field, and a general and robust adaptive reconstruction framework for high-quality images with a wide range of rendering effects. To address complex scene geometry, we propose a novel concept which models the acceleration structure completely implicit, i.e. without any additional memory cost at all, while still allowing for interactive performance. Our contributions advance the state-of-the-art of denoising techniques for realistic image synthesis as well as the field of memoryless acceleration for ray tracing systems.

Kurzfassung

Stochastische Ray-Tracing Methoden sind heutzutage der Industriestandard für realistische Bildsynthese, da sie einen hohen Grad an Realismus erzeugen können, indem sie verschiedene natürliche Phänomene (z.B. globale Beleuchtung, Tiefenunschärfe oder Bewegungsunschärfe) physikalisch korrekt simulieren. Offene Probleme dieser Verfahren sind hohe Rechenzeit für komplexere Szenen sowie Bildrauschen durch unzulängliche Simulationen. Demzufolge sind Beschleunigungstechniken und Entrauschungsverfahren essentielle Komponenten in stochastischen Ray-Tracing-Systemen.

Im Detail präsentieren wir einen interaktiven Filter für globale Beleuchtung in Kombination mit Tiefenunschärfe und einen generischen, robusten Ansatz für die adaptive Rekonstruktion von hoch-qualitativen Bildern mit einer großen Auswahl an Rendering-Effekten. Für das Problem hoher geometrischer Szenen-Komplexität demonstrieren wir ein neuartiges Konzept für die implizierte Modellierung der Beschleunigungsstruktur, welches keinen zusätzlichen Speicher verbraucht, aber weiterhin interaktive Laufzeiten ermöglicht. Unsere Beiträge verbessern sowohl den aktuellen Stand von Entrauschungs-Verfahren in der realistischen Bildsyntese als auch das Feld der speicherlosen Beschleunigungsstrukturen für Ray-Tracing-Systeme.

Summary

Stochastic ray tracing methods have become the industry's standard for today's realistic image synthesis. The popularity of these methods stems from their ability to simulate various natural phenomena of light and camera systems based on physical models. The underlying concept of simulating a series of single light paths through the scene is simple yet general and highly parallelizable. However, the simulation process is a computationally expensive task because larger numbers of light paths need to be simulated for visually pleasant images. If the number of evaluated light paths does not suffice the result will contain noise. While this does not pose a big problem for moderately simple scenarios where advances in consumer hardware already allow interactive or even real-time ray tracing, the problem still remains when complex lighting and camera effects (e.g. global illumination, depth-of-field, or motion blur) are involved. Simulation complexity is also constantly being increased because artists strive for expanding both the geometric complexity of the scene as well as the number and quality of simulated natural phenomena to achieve a supreme degree of realism. Hence, acceleration and denoising are key components in ray tracing systems. Increasing geometric complexity leads to more demanding memory requirements, which is especially problematic for ray tracing because it heavily relies on additional acceleration structures. The memory overhead from these structures is a limiting factor for recent multi-core architectures (e.g. GPUs) which are nowadays often used for accelerating ray tracing, and recent algorithmic design focus on memory-efficient or even memoryless acceleration. In this thesis, we address both issues: image denoising for advanced lighting and camera effects as well as memoryless acceleration for complex scene geometry. In particular, we present an interactive

filter for global illumination in the presence of depth-of-field, and a general and robust framework for adaptive reconstruction from a set of filters based on sparse error estimation for high-quality images with a wide range of rendering effects. To address complex scene geometry, we introduce a novel concept based on geometry presorting which models the acceleration structure completely implicit, i.e. without any additional memory cost at all, while still allowing for interactive performance. The results from our contributions advance the field of denoising techniques for realistic image synthesis as well as research on memoryless acceleration for ray tracing systems.

Zusammenfassung

Stochastisches Ray-Tracing ist heutzutage der Industrie-Standard für realistische Bildsynthese. Die Beliebtheit der Methode resultiert aus ihrer Fähigkeit, diverse natürliche Phänomene von Licht und Kamera-Systemen mittels physikalischer Modelle zu simulieren. Das zugrundeliegende Konzept der Simulation von einzelnen Lichtpfaden durch die Szene ist einfach aber generall einsetzbar und lässt sich hochgradig parallelisieren. Ein Nachteil ist der rechenaufwendige Simulationsprozess, da größere Mengen an Lichtpfaden simuliert werden müssen, um visuell ansprechende Bilder zu erzeugen. Wenn die Anzahl der evaluierten Lichtpfade nicht ausreicht, sind die Ergebnisse verrauscht. Während das für moderate Szenen kein Problem darstellt, da dank des Fortschritts in der Entwicklung von handelsüblicher Hardware für diese bereits interaktives oder sogar Echtzeit-Ray-Tracing möglich ist, bleibt das Problem bestehen, wenn komplexe Licht- und Kamera-Effekte (z.B. globale Beleuchtung, Tiefenunschärfe und Bewegungsunschärfe) beteiligt sind. Die Simulationskomplexität wird kontinuierlich erhöht, da Künstler und Designer danach streben, sowohl die geometrische Komplexität der Szene als auch die Anzahl und Qualität der simulierten natürlichen Phänomene zu steigern, um einen hohen Grad and Realismus zu erzielen. Daher sind Beschleunigungs- und Entrauschungs-Verfahren Kernelemente eines jeden Ray-Tracing-Systems. Die Erhöhung der geometrischen Komplexität führt gleichzeitig zu erhöhten Speicheranforderungen und ist dadurch besonderes für Ray-Tracing ein Problem, welches intensiv auf zusätzliche Beschleunigungsstrukturen aufbaut. Der Speicher-Mehraufwand von diesen Strukturen ist ein limitierender Faktor für aktuelle Mehrkern-Architektur-Systeme (e.g. GPUs) welche heutzutage häufig für das Beschleunigen von Ray-Tracing eingesetzt werden. Daher liegt der Fokus im aktuellen Design neuer Algorithmen auf speichereffizienter oder sogar speicherloser Beschleunigung. In die-

ser Arbeit widmen wir uns beiden Problem: Bildentrauschung für erweitere Licht- und Kamera-Effekte und Speichereffizienz für Beschleunigungsstrukturen für geometrisch komplexe Szenen. Wir stellen einen interaktiven Filter für globale Beleuchtung in Kombination mit Tiefenunschärfe vor sowie ein generisches, robustes Verfahren für adaptive Rekonstruktion basierend auf dünnbesetzer Fehlerabschätzung, welches hochqualitative Ergebisse liefert und beliebige Rendering-Effekte unterstützt. Für das Problem von hoher geometrischer Szenen-Komplexität präsentieren wir ein neuartiges Konzept, welches, basierend auf Vorsortierung der Geometrie, die Beschleunigungsstruktur komplett implizit darstellen kann, d.h. keinen zusätzlichen Speicher verbraucht, aber weiterhin interaktive Laufzeiten ermöglicht. Die Ergebnisse unserer Beiträge verbessern sowohl das Feld der Entrauschungs-Verfahren für die realistische Bildsyntese als auch die Forschung im Bereich speicherloser Beschleunigungsstrukturen für Ray-Tracing-Systeme.

Acknowledgements

Many people supported and inspired me during the work on my thesis. First and foremost, I want to thank all members of the Computer Graphics Lab at the TU Braunschweig which accompanied me throughout the years. This includes all my colleagues, student assistants, the administrative and support staff, and, of course, my supervisor Prof. Marcus Magnor. I'm also grateful for all the students that participated in my seminars, courses, and lectures and I hope they learned as much from me as I did from them. A special thanks goes to all the co-authors of my publications. In particular, I want to thank my colleague and mentor Prof. Martin Eisemann who was (and still is) a constant source of inspiration and support and my colleagues and friends Stefan John and Michael Stengel, who had the mutual pleasure to be my office mates for several years.

Finally, I want to thank my family and my parents to whom I dedicate this thesis ...

Contributions of the Author

In the following, I will clarify my individual contributions to the publications that form my thesis. Even if most of the presented work was conducted by myself, I will follow the scientific rules of conduct and use plural terminology (i.e. "we" instead of "I") throughout the thesis. All projects were supervised by Prof. Dr. Marcus Magnor.

1. Pablo Bauszat, Martin Eisemann, Stefan John, and Marcus Magnor
   Sample-Based Manifold Filtering for Interactive Global Illumination and Depth of Field.
   Computer Graphics Forum, volume 34/1, pages 265-276, 2014.
   In this work, I developed the overall idea with support from Martin Eisemann. My contribution also includes the implementation as well as the evaluation of the proposed method. The paper was co-written with Martin Eisemann. Stefan John participated in the creation of the supplemental video.

2. Pablo Bauszat, Martin Eisemann, Elmar Eisemann, and Marcus Magnor. **General and Robust Error Estimation and Reconstruction for Monte Carlo Rendering.** Computer Graphics Forum (In Proceedings of Eurographics), volume 32/2, 2015. *I developed the general idea together with Martin Eisemann and was responsible for working out the algorithmic details. I performed the implementation and evaluation of the method. The paper was co-written with Martin Eisemann and Elmar Eisemann.*

3. Martin Eisemann, Pablo Bauszat, and Marcus Magnor
   Geometry Presorting for Implicit Object Space Partioning. Computer Graphics Forum (In Proceedings of Eurographics Symposium on Rendering EGSR), volume 31/4, pages 1445-1454, 2012.
   This paper is a joint project of Martin Eisemann and myself. I elaborated the main idea and performed both the CPU and GPU implementation as well as the evaluation of the method. The paper was co-written with Martin Eisemann.

Other authored/co-authored publications that are not included in this thesis:

1. Pablo Bauszat, Martin Eisemann, and Marcus Magnor
   The Minimal Bounding Volume Hierarchy.
   In Proceedings of Vision, Modeling and Visualization (VMV), pages 227-234, 2010.

2. Pablo Bauszat, Martin Eisemann, and Marcus Magnor
   Guided Image Filtering for Interactive High-quality Global Illumination.
   Computer Graphics Forum (In Proceedings of Eurographics Symposium on Rendering EGSR), volume 30/4, pages 1361-1368, 2011.

3. Pablo Bauszat, Martin Eisemann, and Marcus Magnor
   Adaptive Sampling for Geometry-aware Reconstruction Filters.
   In Proceedings of Vision, Modeling and Visualization (VMV), pages 183-190, 2011.

4. Oskar Elek, Pablo Bauszat, Tobias Ritschel, Marcus Magnor, and Hans-Peter Seidel
   Spectral Ray Differentials.
   Computer Graphics Forum (In Proceedings of Eurographics Symposium on Rendering EGSR), volume 33/4, pages 113-122, 2014.

5. Oskar Elek, Pablo Bauszat, Tobias Ritschel, Marcus Magnor, and Hans-Peter Seidel
   Progressive Spectral Ray Differentials.
   In Proceedings of Vision, Modeling, Visualization (VMV), pages 151-158, 2014.

6. Lorenz Rogge, Pablo Bauszat, and Marcus Magnor
   Monocular Albedo Reconstruction.
   In Proceedings of IEEE International Conference on Image Processing (ICIP), pages 1046-1050, 2014.

7. Michael Stengel, Pablo Bauszat, Martin Eisemann, Elmar Eisemann, and Marcus Magnor
   Temporal Video Filtering and Exposure Control for Perceptual Motion Blur. In IEEE Transactions on Visualization and Computer Graphics (TVCG), 2014.

Contents

Part I

Introduction

1

Introduction

1.1 Motivation

Photo-realistic rendering is an elementary goal of image synthesis. Natural phenomena of light and camera systems (e.g. soft shadows, global illumination, depth-of-field, motion blur, diffraction, participating media, etc.) need to be simulated during the rendering process to achieve a supreme degree of realism. Light transport in a scene is realistically modeled by the **rendering equation** which describes the light arriving at a pixel of a virtual image by a series of multi-dimensional integrals [17, 48]. Each integral stands for a specific effect that stems from the light distribution function, e.g. integration over the aperture of the camera for depth-of-field, integration over time for motion blur, or integration over the incoming light direction at a scene point for global illumination. Due to the high dimensionality, complexity, and recursive nature of these integrals, the rendering equation cannot be solved analytically (except for extremely simple scenarios), and solutions are approximated by stochastic point estimation techniques, e.g. Monte Carlo (MC) integration.

Over the last decades, stochastic ray tracing techniques have become an industry standard for this task and are capable of generating photo-realistic images with advanced visual effects. The reason for their popularity is their robustness, scalability, and simplicity of the underlying concept. Point sampling an integral in the context of image synthesis simply means tracing a random light path through the scene to collect the incoming light. The main difference between established methods (e.g. path

tracing [48], bidirectional path tracing [62], Metropolis Light Transport [114], Instant Radiosity [53] and others) is mostly the way these paths are created.

Unfortunately, a major drawback of these methods is their computational load. The reason for this is twofold. Computing a single light path sample requires to perform intersection queries between the scene and the rays forming the path. Often, this computationally expensive task needs to be performed millions or even billions of times for rendering a single image. On top of that, usually thousands of point-samples need to be evaluated for each pixel to reach a high-quality estimate. Insufficient sampling, which is often inevitable due to a limited time budget (especially in interactive and real-time applications), leads to variance in the estimator which manifests itself as noise artifacts in the rendered image. Both issues continue to being challenging because artists strive for increased complexity in both the scene geometry as well as the simulated natural phenomena.

Ray tracing heavily relies on additional acceleration structures which cluster and subdivide the scene geometry to drastically reduce the runtime of ray-scene queries. A common solution for image noise is to apply filtering (or reconstruction, used interchangeably here) to the image or during the rendering process. This biases the result but, if done right, creates a visually more pleasant look for the user than the original noisy image. Acceleration structures and filtering techniques are therefore key components in today's ray tracing systems.

Regardless of whether the application is to generate fast previews for production rendering, interactive tools, or cutting down computation times for movie production, accelerating the rendering process of stochastic ray tracing is an important challenge and has been an area of research for many decades. So far, MC renderers creating photo-realistic imagery are often still limited to offline settings. Recently, interactive ray tracing for moderate scenes has become available on consumer-grade hardware, and the fields of offline and online rendering are gradually growing together. Still, the number of light path samples that can be simulated at interactive or real-time frame rates is still several orders of magnitude too low to provide for fully converged images of sophisticated visual effects.

Recent advances in many-core architectures (e.g. GPUs, multi-processor CPUs, and the cloud) have a major impact on ray tracing, which is highly parallelizable, and has led to significant accelerations [2, 40, 119]. Hence, suitability for multi-core architectures is one important aspect of current algorithm design, and their specifications pose new algorithmic challenges. In the context of acceleration structures this means that memory footprint (which grows with increasing scene detail) gained newly found interest because memory throughput can be a limiting factor on many-core architectures. For filtering design, parallelization becomes a more and more important property wherever fast performance is critical. This is, of course, the case for MC rendering because the tracing of light paths is getting more efficient, and filtering is only desirable if it reduces the noise significantly quicker than additional sampling would do in the same time.

One can summarize the task of speeding up intersection queries and convergence through denoising under the general term of acceleration for MC rendering. While the first one speeds up the evaluation of individual light paths, the second reduces the number of required light paths. Both issues can be seen as orthogonal, and improvements in either area enhance rendering performance as whole.

In this thesis, we focus on two important issues: denoising of stochastic renderings, and memoryless acceleration applicable to large scenes. We propose two new methods for handling image noise. The first method addresses reconstruction of global illumination in conjunction with depth-of-field and is designed for interactive applications. The second contribution proposes a robust and general error estimation technique for adaptive reconstruction based on a given set of filters which excels in its generality and produces high-quality results. Finally, we introduce the concept of *Implicit Object Space Partitioning* for memoryless acceleration which leads to a completely implicit representation of an acceleration structure requiring no memory at all while still allowing for interactive ray tracing.

1.2 Outline

Parts of this thesis have already been presented at the Eurographics conference [3] and the Eurographics Symposium on Rendering [26]. Also, parts have been published in a journal article [4] and a technical report [27]. In the following, we continue by giving a brief overview of the background on the topic of Monte Carlo ray tracing and its acceleration in Chap. 2. We then discuss prior research and present our main contributions in Parts II and III. Part II covers our work on **advanced denoising** and is structured as follows.

- **Chap. 4**
  In this chapter, we present our contribution on **Sample-Based Manifold Filtering for Interactive Global Illumination and Depth of Field** (SBMF). This technique introduces a fast sample-space method to denoise images with global illumination and area light sources in the presence of depth-of-field. Our technique is parallelizable and achieves interactive frame rates on commodity graphics hardware. It does not require any adaptive sampling, yields unprecedented results from very low sampling rates, and it works completely as a post-process. Efficient implementation also makes the algorithm runtime-independent of the circle-of-confusion size and enables efficient support for arbitrarily shallow depth-of-field.

- **Chap. 5**
  This chapter presents our contribution **General and Robust Error Estimation and Reconstruction for Monte Carlo Rendering** (GREE), a general framework for adaptive reconstruction based on error estimation from a sparse set of error estimates. Our contribution is a robust, low-variance selector for choosing the best possible filter per pixel from an arbitrary set of general reconstruction techniques. Because selecting filters per pixel solely on their local expected error results in visible seams and outliers in the final image, our solution formulates the filter choice as a compositing task which is then solved via graph-cuts. Our method is completely generic regarding image content and filtering techniques, and, in contrast to previous approaches, we support arbitrary filter banks with no restrictions on differentiability (even non-filter reconstruction

techniques are applicable). So, most state-of-the-art denoising techniques for image and MC denoising can be utilized. Our approach requires fewer samples than many competitors to achieve higher quality reconstruction of MC renderings. It is orthogonal to fundamental research on image and MC denoising and supports future reconstruction techniques as well.

In Part III, we present a novel approach for **memoryless acceleration**.

- **Chap. 7**
  We introduce the concept of **Geometry Presorting for Implicit Object Space Partitioning** (IOSP) which is an implicit representation of an object partitioning acceleration structure for triangle meshes. The core idea is based on presorting the geometry and we show how to remove any memory requirement by representing the hierarchy as a heap, resulting in an acceleration structure that requires no memory at all: it is represented completely implicit by triangle order. A major benefit of this scheme is that it is easy to parallelize and well suited for many-core processors. We present a parallel construction technique, demonstrating that our approach is applicable to fully dynamic scenes and can render at interactive frame rates.

We conclude the thesis by summarizing our findings and examine promising prospects for future work. To help the reader with the abbreviations and notations used throughout the thesis, we added abbreviation and notation tables in the Appendix.

2

Prerequisites

This thesis partially requires in-depth knowledge of the principles behind image synthesis, in particular stochastic ray tracing and filtering techniques. While we cannot provide a full overview of all topics in detail, we believe that a brief introduction into the different fields eases understanding.

2.1 Light Transport

Rendering Equation We will begin by taking a closer look at the general concept of light transport in image synthesis and the fundamental **rendering equation** from Kay and Kajiya [48]. The rendering equation describes the amount of light that leaves an arbitrary surface point of the scene, denoted by x, towards an outgoing direction, ω_o,

$$L_o(x, \omega_o) = L_e(x, \omega_o) + \int_\Omega f_s(x, \omega_i, \omega_o) L_i(x, \omega_i) \left|\cos \theta_i\right| \mathrm{d}\omega_i \,. \tag{2.1}$$

The amount of light is measured in *radiance* which is given as radiant flux per unit projected area per unit solid angle. Radiance can be distinguished into *incident radiance* $L_i(x, \omega_i)$, i.e. light that arrives at a surface, and *exitant radiance*, $L_o(x, \omega_o)$, i.e. light that leaves the surface point. Radiance has the beneficial property of constancy along a straight line through empty space and can therefore be easily propagated through a scene. This makes radiance one of the most important units for measuring light in rendering, and all point samples in this thesis are radiance samples. While there are other radiometric units of importance, we omit their discussion here and refer the

reader to external literature, e.g. [91]. We also assume a light propagation model based on geometric optics. This means that more complex light characteristics that appear in nature and can be modeled via physical wave optics, e.g. polarization, are out of the scope of the thesis.

The rendering equation is composed of the two parts: the emitted light at that point, L_e, and the light that arrives at x from the sphere (or hemisphere) of incoming directions and is scattered towards the outgoing direction ω_o. The recursive nature of illumination is brought to light by the rendering equation as the incident radiance at x in the integral can be seen as exitant radiance at another scene point towards x. In other words, to evaluate the incident radiance at a point x for only a single direction ω_i, the exitant radiance at the hit point from x towards ω_i need to be computed which again consists of a complete integral evaluation. This illustrates the enormous complexity of light transport and exposes why it is such a computational expensive task.

Surface Scattering The function f_s is called the *bidirectional scattering distribution function* (BSDF) and describes the ratio of light that is scattered from ω_i towards ω_o. It generalizes and usually combines the *bidirectional reflectance distribution function* (BRDF) and *bidirectional transmittance distribution function* (BTDF) which describe the relationship between incoming and scattered light for the upper hemisphere (light reflected from the surface) and lower hemisphere (light transmitted into the surface). The BSDF describes the properties of a surface, i.e. material, and typical, broad classifications are: diffuse, glossy specular, perfect specular, and retro-reflective [91]. Diffuse surfaces scatter light equally in all directions, glossy specular prefer scattering towards a certain direction, perfect specular surfaces scatter *exclusively* towards a single direction, and retro-reflective surfaces scatter preferentially towards the incident light direction. In natural scenes, however, most materials are a combination of the different types.

Separation of Direct and Indirect Illumination If knowledge about the emissive surfaces exist, i.e. the light sources of the scene are known, an alternative representation of the rendering equation separating the lighting into an direct and indirect part can be used:

$$L_o(x, \omega_o) = L_{direct}(x, \omega_i, \omega_o) + L_{indirect}(x, \omega_i, \omega_o) \tag{2.2}$$

with

$$L_{direct}(x, \omega_i, \omega_o) = L_e(x, \omega_o) + \int_A f_s(x, \omega_y, \omega_o) L_i(x, \omega_y) \left|\cos\theta_i\right| \frac{\left|\cos\theta_y\right|}{r^2} \mathrm{d}y \,, \tag{2.3}$$

$$L_{indirect}(x, \omega_i, \omega_o) = \int_{\Omega_{indirect}} f_s(x, \omega_i, \omega_o) L_i(x, \omega_i) \left|\cos\theta_i\right| \mathrm{d}\omega_i \,. \tag{2.4}$$

Here, the direct illumination part integrates over the surface area of all light sources A, where y describes a point on a light source, $|\cos\theta_y|$ the absolute cosine-angle between the normal at y and the direction vector to x, and r denotes the distance between the x and y. Because all direct light is already taken into consideration, the indirect part now only includes light from directions that do not propagate light from emitting surfaces. The benefit of this separation is that the direct and indirect part of the illumination can be evaluated separately. Often, the direct part is more straightforward to compute and other approaches than stochastic ray tracing can be employed.

Measurement Equation To render an image, the radiance that arrives at the camera sensor, i.e. the image plane, needs to be estimated using the rendering equation. This process is mathematically described by the general *measurement equation* which gathers the incident radiance for a pixel by integrating over the 2D domain of the pixel, denoted by P:

$$p = \int_{P \times \Omega} W(x, \omega) L_i(x, \omega) \,\mathrm{d}x \,\mathrm{d}\omega \,. \tag{2.5}$$

Let x be a point on the image plane, W describes the sensor sensitivity, and p describes the total incident radiance, i.e. pixel color in the color space of choice. In this thesis, we continuously use the RGB color space.

Depth-of-Field and Motion Blur More sophisticated camera effects, e.g. depth-of-field (DoF) and motion blur (MB), can be included by additional integration over the 2D lens domain, denoted by U, or the time domain, denoted by T:

$$p = \int_{P \times \Omega \times U \times V} L_i(x, \omega, u, t) \,\mathrm{d}x \,\mathrm{d}\omega \,\mathrm{d}u \,\mathrm{d}t$$

Note that, as the dimensionality of the sample domain increases, each additional dimension complicates the original problem and makes it more complex.

Monte Carlo Integration A multi-dimensional measurement equation in conjunction with the recursive nature of the rendering equation makes an analytical solution infeasible except for very simple scenarios. The problem is additionally aggravated by an integrand with discontinuities (e.g. at geometric edges) and singularities (e.g. at perfect specular surfaces or point light sources). A numerical approach is Monte Carlo (MC) integration which computes a solution using a series of point samples. Over the last decades, it has become a popular option in image synthesis. In general, using MC integration, an integral of the form

$$y = \int_{\mathbb{R}} f(x)\,\mathrm{d}x$$

can be rewritten as a sum of random samples

$$\widehat{y} = \frac{1}{N}\sum_{i=0}^{N}\frac{f(x_i)}{p(x_i)}$$

where $\widehat{y}$ converges to y when $N \to \inf$. Here, the sample values x_i are drawn from a random distribution with a probability density function $p(x)$. It can be shown that the expected value of the estimator $\widehat{y}$ is equal to y, but we will omit further details for brevity and refer to common literature on stochastic and MC integration.

Path Sampling In stochastic ray tracing, MC integration is directly applied to the integrals of the measurement equation and the recursive integral estimation in the rendering equation. Although, they consist of a series of multi-dimensional integrals, it is not required to solve each integral individually. The concept of **path tracing** was introduced by Kay and Kayjiya [48] in conjunction with the rendering equation and computes an image by evaluating multiple random walks, so called *paths*, through the scene for each pixel. Path tracing is a popular rendering technique because it produces unbiased results, is based on a simplistic concept, and can easily be implemented in parallel since each path sample is independent.

We will employ path tracing as underlying rendering method for our experiments. The only required output of the renderer are the radiance values for the path samples and additional features for each path, e.g. geometric information at the primary hit points (the first hit from the camera).

2.2 Acceleration Structures

To generate a random walk through a scene, it is required to repeatedly propagate a ray through the scene and find the closest intersection along the ray's direction. Similarly, for computing occlusions and shadowing of light sources, visibility queries between two scene points are needed. If done trivially, computing the intersection of a ray with the scene would require to check it against each individual scene primitive which is infeasible for complex scenes and anything but efficient. Therefore, ray tracing relies heavily on the use of additional acceleration structures which are built upon the geometry of a scene and accelerate ray-scene intersection queries. While these structures significantly speed up the rendering process, they come with the drawbacks of additional memory demands and the need for rebuilding in dynamic scenes.

Acceleration structures Over the last decades, many acceleration structures (AS) have been proposed for ray tracing and common applications including culling, nearest neighbor searches, and collision detection. An excellent survey can be found in [125]. Generally, AS can be classified in space- or object-partitioning, and hierarchical or non-hierarchical. The most popular ones for ray tracing are kd-trees, grids, and Bounding Volume Hierarchies (BVH) using axis-aligned bounding boxes [52, 101]. While kd-trees and grids subdivide the space of the scene, BVHs partition the primitives of the scene. Unfortunately, all common AS can use a significant amount of memory, especially if complex models are to be rendered. AS are often a major factor in overall memory consumption, requiring typically between ten to twenty percent additional memory, with even higher values reported [64].

Bounding Volume Hierarchies BVHs are frequently the method of choice for interactive and real-time settings due to their comparatively low memory footprint [40, 116], efficient traversal, and fast construction times [37, 83]. Variants of the BVH are integrated in several popular CPU and GPU ray tracing libraries and development kits such as Intel's *Embree* ® [128] or NVidia's *OptiX* ® [87]. The idea behind a BVH is to subdivide the primitives of a scene into possibly overlapping, but optimally distinct sets. For each of these sets a bounding volume – for ray tracing often an axis-aligned bounding box (AABB) – is computed and these are arranged in a hierarchical tree structure. The bounds of every node in this tree are chosen so that it exactly bounds

all the nodes in the corresponding sub-tree and every leaf node exactly bounds the contained primitives. Ray traversal then starts at the root node and if a ray misses a bounding volume in this hierarchy, the whole sub-tree can be skipped. When it comes to memory requirements, BVHs usually have the lowest demands in practice compared to other approaches, as shown in [116]. Also, their memory footprint is known a priori which is a convenient property for construction algorithms. This knowledge stems from the object partitioning scheme which, in contrast to spatial subdivision, references each primitive only once. In this thesis, we focus exclusively on memory reduction techniques for BVHs or built upon their concept of object partitioning.

Part II

Advanced Denoising

3

Background

3.1 Motivation

When images are generated with MC methods and an insufficient number of path samples per pixel, the result is plagued with noise and convergence towards a smooth image is typically slow. Therefore, reconstruction or filtering techniques for MC ray tracing have a long history [76]. The ultimate goal of image reconstruction is to compute the perfect image signal from a noisy estimate. A typical prior in denoising is that the real signal is assumed to be locally smooth and, therefore, combining neighboring information leading to smoother results are actually an improved estimate of the real signal. An alternative view of image reconstruction is that estimates of the rendering equation which are geometrically close (in terms of image or scene distance) show redundancy in their samples and sharing them between the estimates can increase convergence. Consider a point in the scene and light arriving at this point from a certain direction. Probably, the light arriving from that direction from a close-by scene point will be the same as long as visibility or material properties do not suddenly change. Mathematically, a MC estimate expresses the sample mean, i.e. the mean of a set of random samples, and therefore can be considered as a random variable itself with its own mean and variance. Accordingly, the mathematical interpretation of incorporating neighboring samples (or pixels) in a point estimate is to treat them as independent realizations of the same random variable. If done correctly, this can greatly reduce the variance of an estimate, but if inapt samples are included it introduces bias in the estimator. Hence, the goal of filtering for MC ray tracing is to reduce variance to present visually

more pleasing images, while introducing the less bias as possible and doing that efficiently and fast.

Generally, reconstruction techniques can be classified as image-space approaches that operate solely on the pixels of the rendered image or as integrand-approaches that operate in the higher-dimensional space of the scene [77]. While the latter approaches are powerful, they come with the drawbacks that they are computationally costly, often memory consuming, and require to be hooked directly into the rendering pipeline. Therefore, recent research efforts focus on image-space approaches which provide efficient and general solutions, can be applied in a post-processing step, and are founded on solid theoretical background of image denoising. While the underlying concept stems from classic image denoising, working on the input image alone is usually not enough to completely remove MC noise without also losing image details. This is primarily due to the lack of capability to distinguish between noise and scene content. Fortunately, during rendering a lot of additional information are generated and can be used to guide the reconstruction process. This is an important and distinguishing characteristic of illumination filtering in contrast to classic image denoising. Recent research focuses on the design of efficient and fast filters as well as on error analysis-based adaptive reconstruction which is often combined together with adaptive sampling. All methods that we present in this part of the thesis are either image-space or sample-space approaches, i.e. they are working solely on pixels or samples of pixels.

3.2 Related Work

Image filtering is one of the most essential operation in image processing and plays a major role in image smoothing, sharpening, restoration, edge detection, and many other tasks. A desired property of a filter is the ability to preserve edges and many researchers focus on the design of efficient edge-preserving filters (in contrast to elementary linear translation-invariant filters such as Gaussian or Laplacian filters).

Filter Classification Filtering methods are typically classified into *local* (or *explicit*) filters and *gradient-domain* (or *implicit*) methods. Local filters perform convolution

of an explicit filter kernel with the noisy input, while gradient-domain approaches employ (convex) optimization methods to reduce diversity of image gradients. Popular variants of the latter category are based on Total Variation [102] or Weighted-Least-Squares minimization [34]. Gradient-domain methods can compute high-quality global solutions, but they are considered to be computationally expensive and an order of magnitude slower than local filters. While one of the first approaches for MC denoising was based on anisotropic diffusion [72], current research primarily utilizes local filtering.

Bilateral Filtering One of the most popular explicit image filter is the bilateral filter by Tomasi and Manduchi [112]. The bilateral filter extends the standard Gaussian filter by adding a *range weight*, i.e. a weighting term based on photometric properties of the image (e.g. intensity or color). The key idea of this weight is to avoid mixing pixels with divergent photometric attributes, and thus the filter performs edge-preserving smoothing. When applied to images with MC noise, however, the filter tends to over-smooth edges or preserve outliers because the input image is often too noisy to be used for robust weighting. Using a pre-smoothed image for the range weighting can reduce these problems but still sharp edges cannot be preserved [130].

Extensions to the Bilateral Filtering Variants of the bilateral filter replace the range weight from the color image by weights based on additional guide images (*joint* or *cross bilateral filtering* [25, 89]) or use these additional images in conjunction with the color image (*dual bilateral filter* [7]). An issue of bilateral filtering and its variant is that they are computationally demanding and a naive implementation is very slow limiting their applicability. Larger filter windows are commonly required due to heavy MC noise and the runtime per pixel of a filter is a major concern. Unfortunately, the per-pixel runtime of bilateral filtering and its variants in a naive implementation scales quadratically with the window size. Even though various methods for accelerating the original bilateral filter exist [14, 24, 29, 41, 84, 90, 92] they do not seem to be feasible for high-dimensional filtering, are approximative, or cannot be directly used to speed-up joint/cross/dual bilateral filtering.

Patch-based Filtering Patch-based approaches, e.g. Non-local means (NLM) filtering [12] or 3D block-matching (BM3D) [19], extend the concept of bilateral filtering

by using a patch-based distance measurement instead of only considering pixel to pixel distances. Similar to the bilateral filter, patch-based approaches can be extended to compute the weighting from guide images instead of the original color image. While these approaches produce high-quality results and can be considered the state-of-the-art in image denoising, their performance is even slower than the bilateral filter due to the increased complexity.

Geometry-aware Filtering The idea of geometry-aware filtering methods is to detect discontinuities by weights based on geometric properties (e.g. normals, depths, 3D positions, texture albedo colors etc.) instead of noisy color estimates [54, 124]. This approach can be highly beneficial if the guides cover all the image edges well and are not noisy themselves. However, this is not always the case because edges can also be introduced by changes in lighting or visibility, and guides can be noisy due to distribution effects (e.g. from depth-of-field or motion blur). Several research exist which incorporates more sophisticated rendering information to define pixel or sample similarity. [78] proposed the use of a *virtual flash image* as edge-stopping guide image. A virtual flash image is composed of only the direct lighting and perfectly specular light paths based on the insight that these light paths are typically less noisy. While this works particularly well for certain effects (e.g. caustics), issues occur when the direct lighting contains noise (e.g. for area lights or environment lighting) and edges from diffuse or glossy indirect illumination are not covered. *Random parameter filtering* was presented by [106] (and later refined by [85]) and proposes to establish a functional relationship between sample values and their random parameters from the MC process. The method performs joint bilateral filtering in the sample-space, i.e. it computes the filtered pixel value not from nearby pixels but from all the samples of nearby pixels, and adjusts the weight functions based on their dependency on the random parameters. Intuitively, this means that the influence of a feature in the weighting – eventually distinguishing two pixels strongly – is reduced if features of this kind are noisy in the local neighborhood. While this lead to high-quality results even for very low sampling rates (e.g. 8 samples per pixel), performance is slow due to the use of a (joint) bilateral filter in the sample-space. Recently, [22] proposed a generic approach which uses the sample color distributions of pixels to measure similarity in a NLM filter producing high-quality results. The samples of a pixel are stored in a RGB color histogram and histogram distance metrics

are used to compute the similarity between pixels. Drawbacks of this approach are the high memory footprint for the histograms and a high number of required initial samples for robust histogram metrics. It is worth noticing, that none of the above approaches are particularly designed for or target MC denoising in real-time or interactive settings.

MC denoising approaches often have to focus on only a subset of natural phenomena for real-time or interactive performance. Recently, Dammertz *et al.*[21] proposed an approach for denoising global illumination noise based on the edge-avoiding À-Trous wavelet transform [35] using normals, 3D positions and texture albedo as guidance. While edge-avoiding wavelets are extremely fast and give similar results than cross bilateral filter, they contain visually disturbing ringing artifacts. However, this work stands out as it was one of the first that focuses primarily on interactive and real-time MC denoising. Before this thesis, we presented another approach for real-time noise removal for diffuse and moderately glossy surfaces under global illumination in [6]. This work was based on *Guided Image Filtering* (GIF) [45] and employed the normal and depth information as guides. The idea of GIF is to establish a local linear model between one or more guide images and the noisy image, and then compute the filtered output of a linear transformation of the guide. GIF uses linear regression to fit the guide to the noisy data, but only operates in small windows around each pixel so it computes a local solution and not a global one for the whole image. A major benefit of GIF is that the runtime of the filter is independent of the kernel size and a parallel implementation can be done very efficiently. Another advantage is that it is robust to outliers near edges and is not prone to gradient distortion. However, the approach in [6] struggles for scenes including distribution effects such as depth-of-field and motion blur, because it is based on the assumption of noise-free geometric information. Distribution effects pose a major challenge to all geometry-aware filtering approaches that are based on this assumption.

Our first contribution **Sample-Based Manifold Filtering for Interactive Global Illumination and Depth of Field** (SBMF) in Chap. 4 addresses this issue and focuses on interactive removal of global illumination noise in the presence of depth-of-field. This work was published in the *Computer Graphics Forum (CGF)* journal in 2014. In

this work, we employ **Adaptive Manifold Filtering** (AMF) as an approach for efficient high-dimensional filtering. AMF clusters the noisy input signal into parts by splatting them onto several locally smooth manifolds which are created in a recursive fashion. Each manifold is then independently blurred which enforces that information is only shared inside clusters. Finally, the blurred values are weighted and averaged in a final slicing step. The runtime of AMF is also independent of the filter size and it scales linear with the number of used guides which is an important attribute for high-dimensional filtering. We will discuss AMF, its benefits and drawbacks, and our extension in more detail in Chap. 4.

Adaptive Sampling and Reconstruction In contrast to uniform filtering where the same parameters are used for the whole image, adaptive reconstruction methods adjust parameters or kernels locally and can drastically improve the filtering quality. Often, adaptive reconstruction is combined with adaptive sampling [43, 48, 76, 95]. Because no reference image exist – we want to compute it in the first place – it is not trivial to define which filter is optimal. Therefore, researchers have investigated error analysis techniques which try to predict the best sampling and/or filter parameter and several approaches have been proposed over the years. Projecting the problem into another domain can give additional insights into an efficient sampling or reconstruction scheme. Approaches building on this idea are commonly based on the assumption of sparsity within the investigated domain, e.g. the Fourier domain [108], or a Wavelet domain [82]. Axis-aligned filtering [73, 74, 75] performs Fourier analysis on the image and data from the rendering process to estimate the optimal sampling rate per pixel and combines it with adaptive filtering. However, only Gaussian filters are considered limiting the reconstruction quality. Overall, the main drawbacks of frequency analysis methods are that they tend to be computationally demanding and employ slow Gaussian or variants of bilateral filtering.

Faster implementations of edge-preserving filtering are mostly designed for filtering parameters that are uniform and constant over the whole image. Therefore, adaptive reconstruction is simulated by separating the filtering process from the error estimation. Typically, a *filter bank* is created by filtering the noise image multiple times but with different global parameter settings. Then, error analysis is performed and for each pixel the best filter from the filter bank is chosen as final value. The Greedy Error

Minimization [98] restricts the filter bank to Gaussian filters and guides the selection via an approximate bias term and empirical sample variance. In the work of Li et al. [69] and Rousselle et al. [100], Stein's Unbiased Risk Estimator (SURE) [109] is applied for error analysis by exploiting the fact that the MC estimator itself resembles a normal distribution. SURE can estimate the Mean Squared Error (MSE) of a filter for a signal polluted by additive white Gaussian noise if the standard deviation of the noise is known and the filter is (weakly) differentiable. Unfortunately, computed locally for a single pixel, the error estimate can potentially be highly variant. A wavelet-based noise estimator for local filter parameter selection for a single type and varying parameters was presented in [50]. A common drawback of all previously-mentioned approaches is that the variance/MSE estimator is variant in itself, which is why all of them require a subsequent smoothing in a post-process. In consequence, only *semantically similar* filters can be used in the filter bank.

Choosing the *best* technique for every situation is still an open problem, especially, as the choice of an appropriate filter can vary within an image. No robust statistic existed so far that *evaluates and compares* the reconstruction quality of arbitrary filters to make a good choice without knowledge of a reference. Further, even with such a classifier, severe visible artifacts would arise in form of seams wherever filters are switched because filter selection was only considered locally. Our second contribution **General and Robust Error Estimation and Reconstruction for Monte Carlo Rendering** (GREE) in Chap. 5 proposes a general and robust approach to solve both of these problems. This work was presented at the *Eurographics 2015* conference and is published in the accompanying proceedings.

4

Sample-Based Manifold Filtering for Interactive Global Illumination and Depth of Field

Geometry-aware image-space filtering expects the incoming radiance to vary smoothly over the scene surface and works well for moderately glossy and diffuse (indirect) illumination. Hereby, the assumption is that adjacent pixels often represent similar scene points and can be seen as partial solutions to the same integration problem. In the presence of and in conjunction with depth-of-field (DoF) effects, however, this assumption is no longer valid. DoF rendering usually creates different image regions, some are in-focus while others are out-of-focus. In the out-of-focus areas, the area of the scene projected onto a pixel can be significantly larger than for the in-focus areas. Samples inside such pixel need to cover a larger scene area and, hence, their geometric features are often highly variant, i.e noisy. Several image-space filtering approaches, especially the ones aiming at interactive or real-time performance, merge the samples per pixel beforehand to simplify the filtering process. But as soon as samples are merged together, DoF effects can no longer be reconstructed any more as a single pixel may contain information from both in- and out-of-focus areas which are required to be treated separately.

Rendering DoF has been an active area of research, particular in the real-time rendering community which primarily employs rasterization methods. Here, most algorithms to simulate DoF post-process a single pinhole image shot from the centre of

the lens. The result is filtered at each pixel to minimize shading and rendering time [46, 96, 97, 133]. The image created from the lens centre misses important information about hidden surfaces at depth discontinuities. Hallucinating missing details by subdividing the image into depth layers and extending the colors into the hidden areas can lead to overly-blurred results and dark silhouettes [60, 61]. Other approaches make use of the capabilities of modern graphics cards to derive a layered scene representation including the missing information in a single render pass [65, 66]. An image-based ray tracer creates the required samples to simulate the DoF effects, resulting in a solution very close to accurate methods. Having control over the rendering process itself, these methods achieve close to ground-truth results with a minimum of computational effort for DoF. However, these work focus particularly on reconstructing solely DoF, but the situation becomes far more complicated if multiple effects in a MC renderer are to be considered simultaneously.

Prior work for DoF denoising in MC rendering includes the high-quality approach of [15] which can create close to ground truth images with only eight samples but the reconstruction times can be in the range of minutes. Similarly, the problem can be formulated as a layered light field reconstruction problem [113] which can be extended also to support motion blur [80]. Adaptive sampling and reconstruction approaches are often more general in terms of MC noise and can achieve impressive results [67, 68, 69, 99, 106]. However, their runtime prohibit their use in interactive settings as they are either computationally complex or require already a good "initialization" in the form of an increased amount of initial samples. While impressive in their comparably small computational requirements, these techniques are basically complete rendering algorithms on their own and, therefore, cannot be used as a post-process to classic MC rendering which is often crucial for production rendering. In contrast to these approaches, we aim at processing the output of a general MC renderer at interactive frame rates including DoF *and* GI without invasive changes to the rendering algorithm itself. Our goal is to work only on the provided samples to keep the MC renderer unaltered which allows easy integration into existing rendering systems.

Closer to our approach is the *sweep-blur* technique [107] which is a fast sample-based reconstruction filter specifically designed to smooth depth-of-field and motion blur but *without* GI. For each pixel, it conducts a sweep through a fixed neighborhood of samples in front-to-back order and computes the reconstruction kernel of each sample based

on the preceding samples. We adapt this idea to our needs and provide ways to (1) circumvent the costly gathering procedure, (2) include global illumination information, and (3) make it efficient for arbitrarily shallow DoF. Overall, we propose an interactive sample-based filtering approach to handle general distribution effects such as GI, area light sources, and DoF effects in MC rendering. Our technique is parallelizable and achieves interactive frame rates on commodity graphics hardware. It does not require any adaptive sampling, yields unprecedented results from very low sampling rates, and works completely as a post-process. The approach treats the renderer as a black box and operates solely on the output color and geometric sample information. The underlying idea is based on the insight that the GI integral in the rendering equation is an inner part of the DoF integral. Hence, we propose to approximately reconstruct the GI integral of the rendering equation per sample before approximating the lens integral over each pixel. The samples are not merged but splatted on adaptive manifolds, clustering similar samples together before filtering them. This can be seen as a generalization of the adaptive manifolds filter [39] to support arbitrary numbers of samples per pixel which we use for GI reconstruction. Once the GI information per sample is reconstructed, we can handle the DoF by splatting the samples again onto linear manifolds and accumulating their color and influence in front-to-back order for the final result. We employ the concept of the sweep-blur presented in [107] but replace the costly explicit gathering of neighboring samples for each pixel with direct filtering of the manifolds, allowing us to use any fast image filtering technique. This makes the algorithm runtime-independent of the circle-of-confusion size and enables efficient support for arbitrarily shallow depth-of-field. We implemented our algorithm on current graphics hardware (GPU) enabling us to create high-quality images containing GI, area light sources, and DoF effects at interactive frame rates.

The rest of this chapter is organized as follows. We present an overview of our proposed reconstruction filtering method (Sect. 4.1), followed by a detailed explanation of its two parts (Sect. 4.2 and Sect. 4.3). Afterwards, we give information on the implementation in Sect. 4.4 and evaluate our method regarding quality and performance in Sect. 4.5. Finally, we discuss our findings and limitations of our approach in Sect. 4.6.

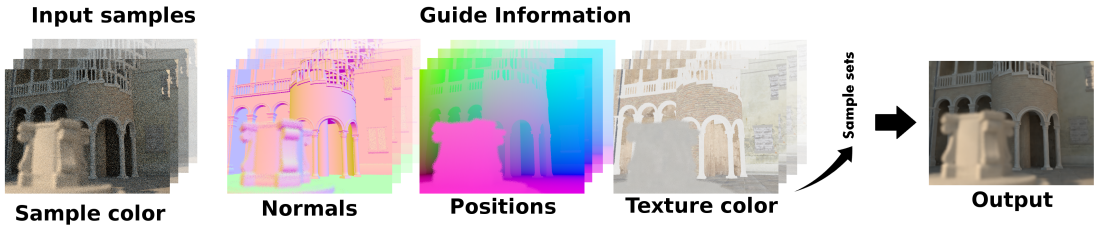


Figure 4.1: Input and output of our sample-based filtering algorithm.

4.1 Method

Input to our algorithm is the sample information produced by an MC renderer, i.e. sample colors, normals, world-space positions, texture base colors, Fig. 4.1. The color of a sample represents the radiance of direct and indirect lighting that leaves the primary hit point of the sample and arrives at the image plane. Our contribution is motivated by the insight that image noise can be classified into two categories:

- noise that is introduced by under-sampling of the integral over incoming radiance at a given scene sample point, and
- noise that is introduced by under-sampling the pixel/lens domain resulting in insufficient sampling of the affected scene regions projected to a single pixel.

The distinct differences in the origin of these noise factors make it difficult to handle both at the same time. However, an interesting observation is that the reconstruction of the integral over the incoming radiance for *each sample* is independent of the DoF noise. This is the key to our approach. Instead of trying to address both sources of noise simultaneously, we can divide the problem into two separate parts and conquer each one independently.

The first step improves the *local* results by filtering over neighboring samples in an edge-preserving fashion, recovering the GI information per sample. Operating on samples instead of pixels increases the complexity of a filter due to an additional dimension in the filtered signal and therefore lead to a performance reduction and poor scaling. For fast and efficient filtering of individual samples without merging their contributions together, we build on the concept of Adaptive Manifolds Filtering [39] and extend it to efficiently operate on separate samples instead of pixels (Sect. 4.2).

The second step of the algorithm addresses the distribution of samples on the image plane for DoF. We start from the concept of the sweep-blur algorithm [107], but replace the costly sample gathering step with a linear manifold filtering step. This step efficiently distributes the samples on the image plane almost linear in runtime to the number of pixels and manifolds (Sect. 4.3) and is independent of the circle-of-confusion sizes of the samples.

4.2 Sample-based Adaptive Manifolds

Our first goal is to reconstruct the GI information for each sample separately. In the following, we first describe the basic concept of Adaptive Manifold Filtering introduced in [39]. We then show how to generalize the idea from pixel- to sample-space.

4.2.1 Background

Filtering based on adaptive manifolds [39] is a fast edge-aware method with attractive properties, e.g. independence on the kernel size and linear scaling with the number of input guides. It can also be seen as a fast approximation of joint bilateral image filtering (JBF) [25, 89]. The main idea is that a non-linear filter becomes linear in a higher-dimensional space of the edge function. In our case, the edge function is *guided* by the input pixel color and other geometric attributes. An adaptive manifold is represented as a regular grid of the size of the input image where each position in the grid represents a certain point in the higher-dimensional space of the edge function. A recursive creation scheme adapts the manifolds to the signal in the higher-dimensional space. This focuses the computational effort on only those regions that contain samples.

Once the manifolds are created, the filter itself proceeds in three main steps: *splatting*, *blurring* and *slicing*. First, the color information of each input pixel is splatted onto the manifolds. The blurring step diffuses the splatted information inside each manifold using a filter that takes the manifold's curvature into account. The slicing step then recovers the final filtered value inside the high dimensional space for each pixel by interpolating the blurred information from all manifolds.

4.2.2 Our Approach

We generalize the adaptive manifolds approach to multiple samples per pixel. In contrast to the original approach where the signal is locally adapted based on clustering the pixel color, we propose to build the manifolds by clustering based on color means which are only computed from a specific fraction of the samples and are adjusted adaptively for each manifold. One major benefit of our approach compared to the adaptive manifolds filter version which uses 3D manifolds (defined by the 2D image domain and the 1D sample domain) is that we support arbitrary, non-uniform numbers of samples per pixel and require less memory as the dimensionality of the guide information is lower.

Let d_N be the number of samples per pixel. We define $s^c(i) = (s^c_1(i), s^c_2(i), \ldots, s^c_{d_N}(i))$ and $s^g(i) = (s^g_1(i), s^g_2(i), \ldots, s^g_{d_N}(i))$ as the vectors holding the sample colors and respective guide information for the pixel i.

Manifold Creation We create the manifolds $\eta^k, k = 1 \ldots K$ in a recursive manner. The first manifold η^0 is created by taking the average of all $s^g(i)$ for each pixel i. In a second step, the manifold is low-pass filtered to ensure approximate linearity across a larger neighborhood of pixels. We then cluster the samples of each pixel based on whether they are "above" or "below" η^0, as described in [39]. We recursively continue to create more manifolds by repeating the averaging and clustering step within each new cluster. After the first manifold, only the samples that belong to the current cluster are used for the average computation. The more manifolds are used, the higher the adaptiveness towards the input samples' guide information. An example is shown in Fig. 4.2.

Filtering Once the manifolds are created, the filtering of the GI information of each sample consists of three steps *splatting*, *blurring* and *slicing*. We splat each $s^c(i)$ belonging to pixel i onto the manifolds while keeping their discretization fixed to the image resolution. Therefore, the splatted value of each pixel i on each manifold η^k is computed as:

$$M^k_{\text{splat}}(i) = \frac{1}{d_N} \sum_{m=1}^{d_N} \phi(\eta^k(i) - s^g_m(i)) s^c_m(i) \quad , \tag{4.1}$$

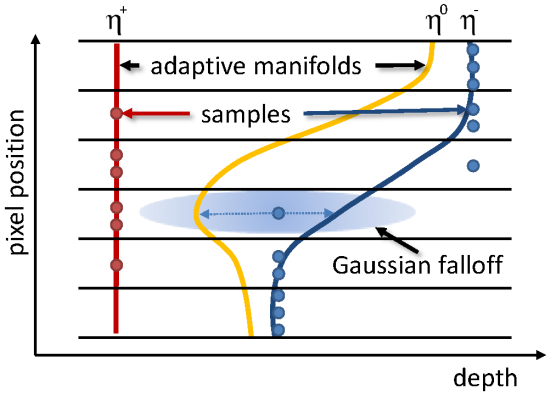


Figure 4.2: Example of the recursive manifold creation scheme using only depth as guide information. The signal of the manifolds quickly adapt to the input samples' depth values. Filtering is then performed on each manifold separately before the slicing step.

where ϕ is a Gaussian kernel used to control the falloff of the splatting. In addition, the accumulated splatting weight per manifold and pixel is saved as

$$M^k_{\text{splatweight}}(i) = \frac{1}{d_N} \sum_{m=1}^{d_N} \phi(\eta^k(i) - s^g_m(i)) \quad . \tag{4.2}$$

To compute the blurred manifolds M_{blur} and $M_{\text{blurweight}}$, we adopt the approach from [39] and apply the Domain Transform filter [38] which respects the manifold's curvature and apply it to the color information on the manifold. The final filtered result $G_m(i)$ for each sample $s^c_m(i)$, and therefore the recovered outgoing radiance, is computed during the slicing step according to

$$G_m(i) = \frac{\sum_{k=1}^{K} w^k_m(i)\ M^k_{\text{blur}}(i)}{\sum_{k=1}^{K} w^k_m(i)\ M^k_{\text{blurweight}}(i)} \quad , \tag{4.3}$$

$$\text{with} \quad w^k_m(i) = \phi(\eta^k(i) - s^g_m(i)) \quad . \tag{4.4}$$

$w^k_m(i)$ can be stored for each sample during the splatting to avoid recomputing it again in the slicing step.

4.3 Fast Sweep-Blur

Given the reconstructed GI information for each sample from the sample-based adaptive manifolds filter (Sect. 4.2), we apply a second filtering step to simulate the depth-of-field

effect. Our approach is based on the sweep-blur in [107] but improves performance by several orders of magnitude for shallow depth-of-field as our approach is independent of the size of the circle-of-confusion (CoC). We first describe the basic concept of the sweep-blur before explaining our extension.

4.3.1 Background

To emulate the DoF effect the sweep-blur algorithm proceeds as follows. For each pixel i, all samples within a certain neighborhood are collected, sorted in depth, and processed in front-to-back order. The idea is that "each sample should be a combination of its denoising filter and the denoising filters of the samples in front of it" [107]. Samples in focus tend to use a small reconstruction filter, but if more samples are in front of it, a wider filter will increase the focus sample's influence. Vice versa, a blurry background pixel tends to use a wide reconstruction filter but if sharp samples are in front of it, the background sample's contribution is diminished. More details can be found in [107]. To speed up the sorting, the samples can be splat onto layers, or linear manifolds. This removes the need for explicit depth-sorting, but the samples still have to be gathered for each pixel. Thus, the original algorithm does not scale well with the size of the neighborhood and shallow DoF is problematic.

4.3.2 Our Approach

We reformulate the sweep-blur algorithm as a fast filtering process similar to Subsec. 4.2.2. We create n_L linear manifolds, consisting of the focal plane itself and n_F linear and equidistantly distributed manifolds in front and n_F behind the focal plane ($n_L = 2n_F + 1$). These manifolds represent different CoCs ranging from zero to a user-defined maximum size τ. Alternatively, τ can be set to the samples' maximum CoC. Each manifold is associated with a filter which support resembles the size of the manifold's associated CoC, Fig. 4.3.

Each manifold stores an RGBα value for each pixel. The RGB-value contains the accumulated color value of its assigned samples. The α-channel stores the number of associated samples divided by d_N and therefore sums up to one over the manifolds. Blurring an individual manifold simulates the process of distributing the associated samples in the image domain w.r.t. their CoC. During this process the α-channel

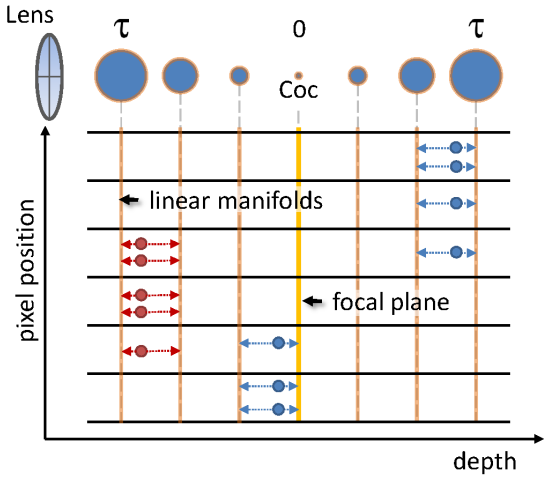


Figure 4.3: Example of linear manifolds for the fast sweep-blur algorithm. Each manifold is orthogonal to the image plane and has an associated circle-of-confusion size. Samples are splatted according to the two closest manifolds.

represents the number of samples in the filter support and by dividing with the alpha-channel afterwards (similar to the concept of homogeneous coordinates) the correct result is computed even in the presence of uneven sample distributions or missing samples.

Splatting & Blurring In a first step, each sample is assigned to the manifolds based on its signed CoC. In contrast to the splatting on the sample-based adaptive manifolds (Subsec. 4.2.2) the influence of each sample is linearly interpolated among the two closest manifolds based on their distance to the respective manifolds. This ensures total conservation of a sample's energy. Each manifold is then filtered with the same filter bank consisting of a series of n_F simple filters (e.g. box or Gaussian) with filter support ranging from 0 to τ. Although a manifold is only associated with a single filter scale f^k, potentially all blurred manifold versions are needed during the accumulation step. The choice of filter is directly related to the Bokeh effect that should be achieved (see Sect. 4.4 for discussion).

Accumulation The manifolds are then processed in front-to-back order, and each manifold's contribution to the final image is computed. Ideally, each manifold contributes only its associated, weighted samples filtered with f^k. However, due to the

fact that the filter scale of a sample should be adjusted based on the filter scales of preceding samples, the actual contribution of a manifold is given by a weighted RGBα average of all the filtered versions of the manifold. We denote the contribution of the k-th manifold (with $k = 1 \dots n_L$) as A^k and its filtered versions by B^k_f (with $f = 1 \dots n_F$). The contribution of a pixel i on the k-th manifold is then computed by

$$A^k(i) = \left(\sum\nolimits_{f=1, f\neq f^k}^{n_F} B^k_f(i) \cdot cov^k_f(i)\right) + B^k_{f^k}(i) \cdot cw^k_{f^k}(i) \tag{4.5}$$

Note that a separate weighting is applied for the filter scale f^k that is associated with the k-th manifold.

The weight cw^k_f for a filter scale f defines for each pixel how much of the blurred manifold B^k_f should be used. It is initially assumed to be one, but is reduced depending on the per-pixel coverage of all the other filter scales in the preceding manifolds. For each pixel i in the manifold k, this can be expressed by

$$cw^k_f(i) = 1 - \sum_{l=1, l\neq f}^{n_F} cov^k_l(i) \tag{4.6}$$

where cov^k_f denotes the sample coverage value. Intuitively, this value can be understood as the fraction of samples that were distributed into the domain of a pixel from the samples of the pixel itself or from neighboring pixels by filtering the manifolds preceding to the k-th manifold with the filter scale f. It is computed as the sum over all filtered, weighted α values of the preceding manifolds

$$cov^k_f(i) = \sum_{m=1}^{k-1} (B_\alpha)^m_f(i) \cdot cw^m_f(i) \tag{4.7}$$

For a better understanding of the recursive behaviour of Eq. 4.6 and Eq. 4.7, note that for the first manifold, $k = 1$, the coverage value will always be 0 and, therefore, the weighting term $cw^1_{f^1}(i)$ will always be 1. The first manifold will always contribute its samples solely filtered with its associated filter scale f^1. The second manifold will then compute its contribution as an interpolation between its blurred versions filtered with filter scale f^1 and f^2, giving higher weights to f^1 in regions already obscured by the first manifold. The process will then continue through all the manifolds and w.r.t. all filter scales.

(a) **(b)** **(c)**

Figure 4.4: Close-up of a scene filtered with 15 adaptive manifolds. (a) Samples that are under-represented on the adaptive manifolds cause inaccuracies due to floating-point imprecision (see the black halos around thumb and wand). (b) Leaving these samples unfiltered causes disturbing outliers as the pixel colors do no longer match those of neighbouring pixels. (c) Replacing sample color with an average of the input samples in the region gives smooth visual results.

Finally, the final pixel color $\mathbf{F}(i)$ is then computed by normalizing the accumulated layer contributions

$$\mathbf{F}(i) = \frac{\sum_{k=1}^{n_L} A^k(i)_{RGB}}{\sum_{k=1}^{n_L} A^k(i)_{\alpha}} \quad . \tag{4.8}$$

The sample coverage values can be computed efficiently by storing the coverage for all filter scales and simply adding the weighted coverage of the current manifold's filter scale during the front-to-back iteration. Mathematically, the result is identical to [107] with the benefit that the per-pixel reconstruction filter is now applied to each manifold at once which can be implemented very efficiently.

4.4 Implementation

In this section we give some further details for re-implementation.

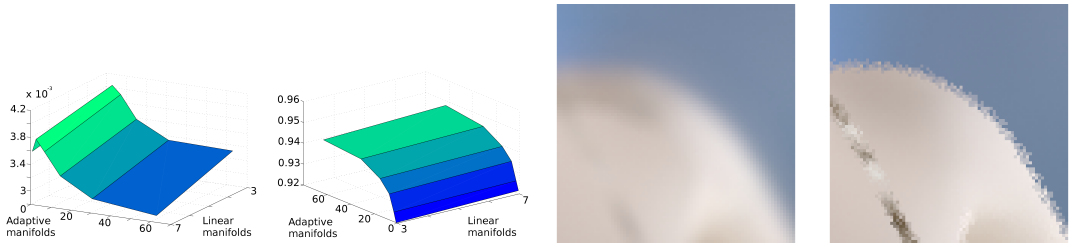
Sample-based Adaptive Manifolds When the number of adaptive manifolds is low and/or the Gaussian distance falloff is very large, it can happen that samples are not well represented by any manifold, causing the denominator in (4.3) to become very small which leads to numerical issues. Such samples require special treatment. If the sample is discarded completely, information is lost which can be disadvantageous for low sampling rates. Simply using the unfiltered, original sample color, on the other

hand, can lead to visual outliers. To avoid increasing the number of manifolds, we solve this issue as follows: when the denominator in (4.3) falls under a user-defined threshold for a sample, we set its color to the pixel color of the noisy input image blurred with a small box filter. We found a radius of 4-8 was sufficient for our test scenes. Fig. 4.4 shows the effect of these outliers and the visual result of our approach.

Fast Sweep-blur Besides the sample-based adaptive manifold filter, also the fast sweep-blur for the sample distribution can directly be implemented on GPUs. For the assignment of samples to the manifolds, all pixels of the image can be processed in parallel. The blurring of the sample values on the linear manifolds depends on the choice of reconstruction filter for the desired Bokeh effect. For efficiency reasons we concentrated only on simple box and Gaussian filters which both can be implemented on the GPU very efficiently in $O(n)$. This is different to the original sweep-blur algorithm which is roughly independent of the desired Bokeh effect, but orders of magnitude slower than our approach. For the box filter implementation (which was also used to generate the results in Sect. 4.5), we build a summed-area-table (SAT) [18] once per manifold and then re-use the SAT for computing arbitrary filter scales for that manifold. Similar to [107], we found in our experiments that seven manifolds (three for front/back out-of-focus regions and the focal plane) produce sufficient results for moderate DoF.

Memory consumption Because the manifold dimensions of our proposed algorithm does not change in comparison to the original image denoising approach, the memory footprint of our GI filtering step is similar to the one reported in [39]. We only need to store an additional buffer which holds the average of all the samples in the current manifold cluster during the generation of a manifold.

For the second step, the linear manifolds are processed consecutively. For each manifold we store an RGBα image with the splatted samples and filtered versions of this image (one for each filter scale). Additionally, we store floating-point coverage images for each filter scale which accumulates the coverage values during the manifold processing. For fast box filtering, a single additional buffer for storing a RGBα SAT is needed.



(a) MSE **(b)** SSIM **(c)** **(d)**

Figure 4.5: Effect of the number of layers on the final image quality of the Fairy Forest scene. The (a) MSE and (b) SSIM are plotted for combinations of 1, 3, 7, 15, 31 and 63 adaptive manifolds for GI filtering and 3 and 7 linear manifolds for DoF filtering. (c) Too few adaptive manifolds (1 adaptive / 7 linear) results in over-blur due to the loss in feature discrimination along the manifolds. (d) With too few linear manifolds (63 adaptive / 3 linear) too many samples are splatted onto the focal plane preventing the DoF filtering.

4.5 Results

We have implemented our filtering technique on a GPU using CUDA and evaluated it for different MC rendered scenes. We use a uni-directional path tracer to create the initial samples. For all test scenes we use three bounces of indirect illumination and explicit sampling of light sources with a single sample per path vertex. All statistics were measured for an image resolution of 1024×768 pixels on an Intel Core i7-2600, 3.40 GHz and 16 GB RAM PC with an NVIDIA GeForce 780 GTX running Windows 7 64-bit.

Parameter Evaluation We evaluated the mean squared error (MSE) [126] and structural similarity index (SSIM) [127] for different numbers of manifolds for both the GI filtering step and DoF filtering. The results are plotted in Fig. 4.5. The image quality is logarithmically correlated with the number of adaptive manifolds. As the time needed for filtering is almost linearly correlated to the number of manifolds, choosing 15 adaptive manifolds seems to be a good trade-off between speed and visual quality. Currently, our implementation only supports to choose between three and seven linear manifolds for the DoF filtering step. Using only three results in objectionable noise for slightly out-of-focus region as the samples are assigned to the in-focus layer. Increasing the number of layers to seven creates visually more convincing results and also reduces the MSE by another 4–12%.

Quality comparison In Fig. 4.7 we compare our method with real-time image-space filtering approaches for GI renderings [39] and the sweep-blur algorithm [107]. While classic image-space filtering struggles in the transition regions from in- to out-of-focus, our method denoises all regions of the image evenly. Of course it must be noted that the sweep-blur was not designed to handle GI noise, while the image-space filtering technique was not designed to handle DoF. We omit a direct image comparison of our fast sweep-blur with the original [107] as the results are visually equivalent. To the best of our knowledge, our proposed approach is the first interactive post-processing method for MC renderings to handle both effects simultaneously.

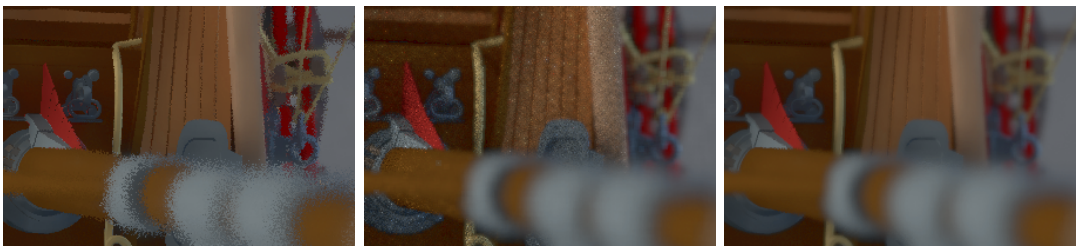
In Fig. 4.8 we compare the noisy input images and the filtered results of our test scenes with reference images rendered with 4096 samples. The MSE as well as the SSIM show that our filtered results are a drastic improvement in comparison to the noisy input and are close to the reference rendering, even though only four samples per pixel are used. More examples can be found in the accompanying video.

We compare our method with two high-quality offline filtering methods which entwine adaptive sampling with image reconstruction, the Robust Denoising method of Rousselle et al. [100] and the SURE-based optimization presented in [69]. Fig. 4.6 shows the output of the two methods with the default parameters and the implementation given by the authors for four samples per pixel compared to our result. Because these methods rely on adaptive sampling for general MC noise removal, they are not designed for low sampling rates and we additionally show their results for 32 samples per pixel. The MSE (with 4spp) using our technique (MSE=0.0031) is slightly higher than with Robust Denoising (MSE=0.0016) or SURE (MSE=0.0021) but the artifacts are perceptually less disturbing. Our results are somewhat more blurry but less prone to visible outliers and noise.

Performance results In Fig. 4.9, we report the performance of our filter method in milliseconds for different sampling rates and numbers of adaptive and linear manifolds. Our algorithm spends most of the time for the local sample reconstruction step, especially when a larger number of adaptive manifolds is used. Note, however, that when the number of samples is quadrupled from 4 to 16, the filter only takes approximately twice as long. This improved scaling is due to the relaxed dependency on sampling rate, as the blurring step is independent of the number of samples.

(a) Reference **(b)** Ours (4 spp) **(c)** Robust (4 spp) **(d)** Robust (32 spp) **(e)** SURE (4 spp) **(f)** SURE (32 spp)

Figure 4.6: Comparison to state-of-the-art offline techniques requiring up to several minutes of computation. From left-to-right: (a) Reference image with detail region marked in yellow, and detail insets for (b) our solution with 4 samples per pixel (spp), Robust Denoising [100] with (c) 4 and (d) 32 adaptive spp, and the SURE-based optimization in [69] for (e) 4 and (f) 32 adaptive spp.



(a) **(b)** **(c)**

Figure 4.7: The figure shows a close-up comparison between (a) classic image-space filtering approaches that cannot handle DoF effects [39], (b) the sweep-blur algorithm that handles DoF but no GI [107] and (c) our method.

	3 manifolds	7 manifolds
Sweep-Blur (radius 4)	29,49	252,35
Sweep-Blur (radius 8)	108,23	983,40
Sweep-Blur (radius 12)	230,73	2095,43
Sweep-Blur (radius 16)	352,23	-
Ours (all radii)	3,62	14,55

Table 4.1: The table shows the performance in milliseconds for different filter sizes of the original sweep-blur implementation and our improved method for four samples per pixel.

A more detailed analysis of the performance is given in Fig. 4.10. Here, the individual times in milliseconds for the sample-based manifold filtering using 15 adaptive manifolds and the fast sweep-blur using 7 linear manifolds are shown. Currently, the clustering and creation of new manifolds is the bottleneck. Note that the blurring phase in the local reconstruction step is independent of the actual sampling rate. During the image plane distribution step, only the classification of the samples on the linear manifolds scales linearly with the number of samples per pixel.

Finally, we reimplemented the advanced layered version of the original sweep-blur algorithm proposed in [107] on the GPU and compare it to our approach which avoids the explicit gathering of samples per pixel (Table 4.1). We give performance numbers in milliseconds for four samples per pixel and 3 and 7 linear manifolds. Notice that the runtime of our method is independent of the used filter sizes.

4.6 Discussion

The current performance-limiting factor of our algorithm is the local sample improvement for GI (Sect. 4.2). Although the number of manifolds can be reduced to achieve better performance, it usually takes up to 15 to achieve sufficient edge preservation in the image sample signal. Image quality in the in-focus regions heavily depends on the condition of the guide information that is available. While we only use normals, world-space position, and texture base color, one could also use more information such as secondary hit point geometry [42], surface roughness or even ambient occlusion factors as shown in related work [106]. With a higher number of input samples, even the

sample color itself could be used as a guide to preserve high-frequency GI effects not represented by other guides.

Using our presented approach, the computational overhead introduced by working on samples instead of pixels is only loosely related to the number of manifolds. The actual dimension of each manifold is unaffected by the number of samples but instead remains equal to image resolution (c.f. [39]).

It might be possible to speed up the GI filtering step by using an enlarged output image where each pixel in an $n \times n$ block represents a single sample and is later combined for an output pixel of the final view. This would allow one to directly apply the original Adaptive Manifolds filter [39] but comes with several drawbacks. It is unknown how to filter such an image if not exactly $n \times n$ samples are available which would create pixels which do not contain any information, and secondly, it is unclear how to incorporate adaptive sampling into such a scheme which is possible in our approach without any further changes.

Even with a sufficient number of manifolds artifacts may appear. Smaller radii keep details such as hard shadow edges, but the GI noise will appear as visible splotches on the surfaces that result in an unsteady display in animations. A larger radius results in smoother overall appearance which is visually more pleasing but introduces too much bias and removes high-frequency GI effects (e.g. contact shadows). Separating the computation of direct and indirect lighting, as it has been done in [6], could potentially deal with these kinds of artifacts and could be easily used in this approach as well.

Another limitation is that the filter tends to over-blur some image regions. The reason for this is twofold. First, the samples in out-of-focus regions are essentially blurred twice during the GI reconstruction step and during the splatting in the image domain, although it is assumed that out-of-focus regions do have less corresponding samples and are therefore blurred less. Second, using too few linear manifolds for the splatting in the image domain causes in-focus samples (with a small circle-of-confusion) to be splatted over a larger window than the one they actually represent. The latter can be handled by increasing the number of linear manifolds or by improving their position (e.g. by placing them with respect to the circle-of-confusion distribution of all the samples).

We primarily focused on DoF denoising for out-of-focus objects viewed directly from the camera. DoF in reflections is not handled specifically. As we base our filter

on scene information at the primary hit point, reflections, in general, are blurred in the first filtering step. Hence, noisy DoF artifacts in reflections seldom appear at all. A good example can be seen on the floor in the PIRATES scene in Fig. 4.8. Adequate fast filtering of reflections is generally an open problem that deserves further attention.

Our contribution in this chapter operates with global parameters on the whole image and, thus, performs uniform reconstruction. We will address adaptive sampling for MC denoising in the next chapter with a general and robust framework. The reader is encouraged to keep in mind that the method from this chapter could also be used in conjunction with this framework.

(a) Ship - *Noisy* (MSE=0.0117, SSIM=0.8375) — *Filtered* (MSE=0.0024, SSIM=0.9234)

(b) Pirates - *Noisy* (MSE=0.0197, SSIM=0.7699) — *Filtered* (MSE=0.0014, SSIM=0.9612)

(c) Fairy Forest - *Noisy* (MSE=0.0393, SSIM=0.6120) — *Filtered* (MSE=0.0033, SSIM=0.9524)

Figure 4.8: Comparison of input image (left, rendered with four samples per pixel), the output of our approach (middle, using 15 manifolds for the local reconstruction and 7 manifolds for the image distribution step) and the reference rendered with 4096 samples per pixel (right). The MSE and SSIM values for the noisy and filtered images are given with respect to the reference images.

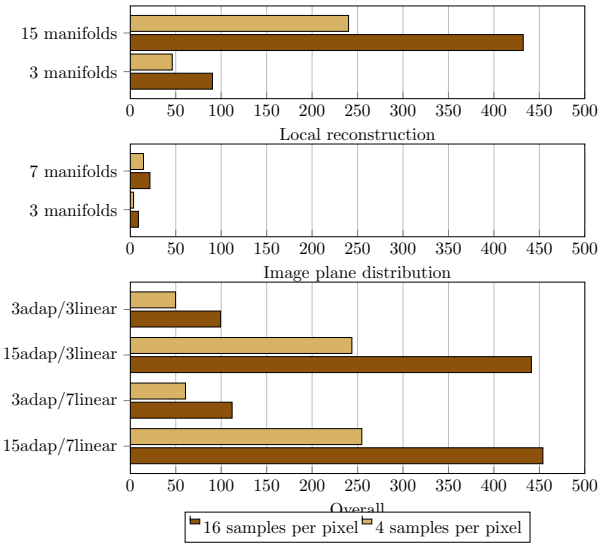


Figure 4.9: The figure shows the performance of the two filtering steps for four and 16 samples per pixel and various numbers of adaptive and linear manifolds. All values are given in milliseconds.

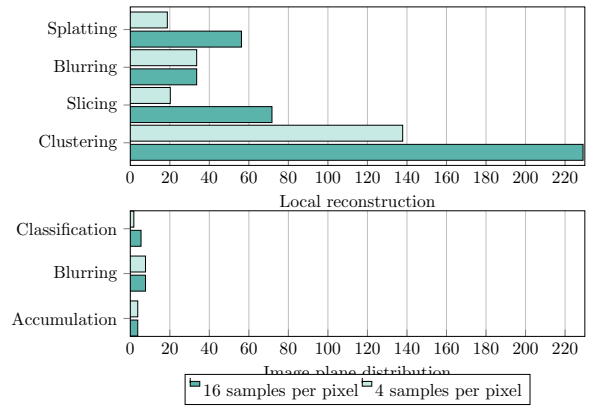


Figure 4.10: The figure shows detailed timings in milliseconds for the filtering process (including SAT generation) for four samples per pixel using 15 adaptive manifolds for the local reconstruction and 7 linear manifolds for the fast sweep-blur.

5

General and Robust Error Estimation and Reconstruction for Monte Carlo Rendering

As the distribution of MC noise is usually spatially-varying – due to changes in geometry, lighting, or secondary effects – choosing an optimal reconstruction filter adaptively per-pixel can significantly increase the filtering quality compared to uniform denoising. Adaptive reconstruction pose two major challenges: determining the optimal filter per pixel without reference and efficient computation of edge-ware filtering with local varying kernels. Recent approaches make use of a *filter bank*, a set of reconstructions using filters with different parameter settings [50, 69, 98, 99, 100], from which one result is chosen per pixel to emulate spatially-varying and anisotropic kernels. This has the advantages that fast filter implementations designed for global parameters can be used to construct the filter bank and that the error analysis-based selection is separated from the reconstruction. Up to now, this selection has relied on risk estimators which suffer if input variance is high, or general image-noise classifications, which cannot robustly distinguish between noise and high-frequency image content.

Our contribution in this chapter is a robust, low-variance selector for choosing the best possible reconstruction per pixel from an arbitrary set of general reconstruction techniques, based on two key observations. First, noise distributions within a small local window, although spatially-variant, change rather gently in most parts of a natural image. Thus, the reconstruction error of a filter candidate is in general locally smooth

across the image and can be well approximated via an interpolation of sparse precise error estimates at carefully chosen locations within the image plane. Second, a suitable reconstruction for a low sample count is more effective than a mediocre reconstruction for a high sample count. Consequently, an algorithm to choose the best input from a given filter bank can be more important than spending computing time on higher sampling rates. Actually, for many samples, a suitable reconstruction is more important due to the convergence rate of the MC estimator and potential bias introduced in the reconstruction process.

We leverage these observations by reducing the number of samples spent on the noisy MC image and redistribute the remaining samples to create *filter caches*, which are highly sampled sparse image locations with strongly reduced variance. These filter caches serve as a robust, sparse error estimation for *any* reconstruction technique, and we can produce a dense error estimation via interpolation. Related to cross-validation techniques in statistics [59] where samples are removed to validate the model fitness, the filter cache samples do *not* contribute to the filtering process. They are solely used to validate the fitness of the filters in the filter bank. Using fewer samples outside the filter caches leads to more variance in the filter input but the variance reduction due to the improved filter selection largely outweighs this downside.

Nonetheless, selecting filters per pixel solely on their local expected error will result in visible seams and outliers in the final image. Ad-hoc solutions, such as smoothing the filter-selection map, are only possible if the filter bank entries are semantically related, e.g. if they represent different parameter choices of the same filter. However, it becomes a challenge to support arbitrary filters within the filter bank. Such problems are known from image-compositing tasks, such as panorama stitching [111], digital photo-montage [1] or image-based rendering [9]. Gradient-domain compositing removes color shifts between input images [88] but come at the cost of a potentially different bias. One very successful approach is to formulate compositing as a labeling problem, which can be solved efficiently using graph cuts [11]. We will adopt this approach to our needs and formulate seamless filter selection as a compositing task to fuse different filters and to optionally incorporate the radiance values of the filter caches into the final result.

Our method is completely generic regarding image content and filtering techniques, as long as our two key assumptions hold. Hence, in contrast to previous approaches, we

support arbitrary filter banks with no restrictions on differentiability (even non-filter reconstruction techniques are applicable) and most state-of-the-art denoising techniques for image and MC denoising can be utilized. The used filters do not need to be semantically related in any way, e.g. that neighboring indices in the filter bank have to refer to similar sizes, etc. The only requirement is that all filters in the filter bank operate on the same input, e.g. some filters may only be applicable to low dynamic range images while others operate on the original high dynamic range radiance values. We will show that our approach requires fewer samples for higher quality reconstruction of MC renderings than many competitors. It is orthogonal to fundamental research on image and MC denoising and will support future reconstruction techniques as well. As mentioned before, our previously presented filtering techniques could potentially be used to create a filter bank for the framework that we will propose in the following. Finally, it is worth noticing that any MC effect can be used and the reconstruction quality solely depends on the filter bank.

The rest of this chapter is organized as follows. First, we will give a more detailed motivation for the insights of our method before we describe our framework, Sect. 5.1. Afterwards, we will show the results of our framework and compare it to other state-of-the-art adaptive reconstruction methods in Sect. 5.2. We will discuss our findings in Sect. 5.3.

5.1 Method

Our algorithm is based on several insights, which we will illustrate in the beginning of this section for a better motivation. First, a good choice for a reconstruction filter is often more beneficial than increasing the number of samples. Second, it is possible to make coherent filter choices in many regions of the image without introducing a large error. This property is key to interpolating filter error estimates and will allow us to avoid seams due to filter changes.

Optimal Filter Selection vs. Sampling Rate To show that in many cases it is more beneficial to choose appropriate filter settings instead of using a higher sampling count with mediocre reconstruction, we compare two different reconstruction techniques using the same filter bank. The filter bank consists of four Gaussian filters

	Opt. (16 spp)	Opt. (32 spp)	SURE (32 spp)
Conference	2.344	1.605	12.327
Sibenik	0.258	0.157	0.758
Toasters	0.156	0.096	0.187
Sanmiguel	9.831	6.419	16.880

Table 5.1: The table shows the MSE (10^{-3}) for the Sibenik, Conference, Toasters ans Sanmiguel scene for 16 and 32 spp and reconstructed from a set of filter using optimal selection (Opt.) or SURE selection.

with $\sigma_{\text{domain}} = [2, 4, 8, 16]$ and four joint-bilateral filters with $\sigma_{\text{domain}} = [1, 2, 4, 8]$. Normals, world-space positions, and texture albedo colors are used as joint guides with $\sigma_{\text{normal}} = 0.8$, $\sigma_{\text{position}} = 0.6$, and $\sigma_{\text{texture}} = 0.25$. As a reference, we employ the SURE estimator from [69] with 32 samples per pixel (spp), which is one of the current top-ranking selection techniques. For comparison, we test a reconstruction with 16 spp, for which we always chose the most optimal filter (determined by comparing to a reference image with 20000 spp). Further, we also tested an optimal reconstruction with 32 spp to determine an upper limit of the reconstruction quality. Table 5.1 depicts the MSE for several test scenes.

The optimal filter selection with just 16 spp reduces the error up to 81% (51% on average) compared to SURE using twice the number of samples. With an equal amount of samples the optimal filter selection reduces the error up to 87% (69% on average). This finding illustrates the potential and importance of a good filter selection procedure.

Coherent Filter Selection Fig. 5.1 shows color-coded visualizations of the squared error for the Gaussian filter, the joint-bilateral filter [25, 89], non-local means filtering [12], the BM3D denoising [19] and the BLS-GSM filtering method [93] for the Sibenik scene and 16 spp. Additionally, the squared error for Guided Image Filtering [45] using varying radii is shown. We observe that the error of a filter varies rather smoothly for most regions of the image. An additional observation is that in many regions the error for more than one filter is close to the optimum, which is interesting because it shows that the filter selection is not always explicit. Instead, multiple candidates can be considered near to optimal.

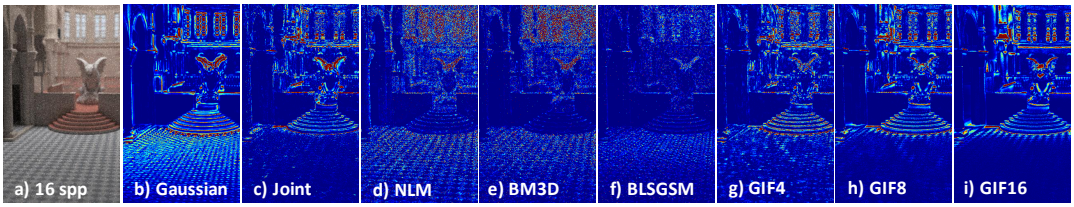


Figure 5.1: Error visualization for the SIBENIK scene with 16 spp (a) for different types of MC denoising filters and parameters. The first 5 insets show the squared errors for the (b) Gaussian ($\sigma = 8$) filter, (c) the joint-bilateral filter ($\sigma = 8$), (d) non-local means filtering (window size=16), (e) BM3D denoising ($\sigma = 0.8$) and (f) the BLS-GSMfilter ($\sigma = 0.06$). The insets (g)–(i) show the squared errors for the Guided Image Filter with varying radii (4, 8, 16).

We examine the impact of a coherent filter selection compared to an optimal selection. Here, we deliberately chose non-optimal filters from the filter bank to enforce spatially-consistent filter choices but restricted the overall solution to have a defined maximum error.

Fig. 5.2 shows three filter selection maps with varying coherency for the SANMIGUEL scene using a similar filter bank as in Table 5.1. Each color represents one entry in the filter bank used for reconstruction of the final image. The optimal per-pixel selection (on the left) reduces the error to 8.0% of the MSE of the noisy image. Coherent selections still result in a low overall error of 8.4% (middle) and 9.1% (right). Similar observations have been made with other test scenes (see supplemental material for more details).

It shows that the optimal filter selection map is comparably noisy, but in large regions, the filter selection can be made coherent without introducing significant errors. However, for specific regions the filter selection is indeed crucial. These findings imply, that a filter selection focusing only at sparse, but carefully chosen pixel locations can be sufficient for a good filter selection across the whole image.

Based on the previous insights, we introduce a filter-selection process for a general filter bank in order to benefit from a better reconstruction, which uses the following input: A noisy MC rendering $\mathbf{N}$ computed from a user-defined number of samples per pixel, a filter bank $\mathbb{F}$ consisting of the results $\mathbf{F}_0, \ldots, \mathbf{F}_m$ of different reconstruction techniques (filtered images) using $\mathbf{N}$ as input, and access to the renderer itself to compute additional samples to produce filter caches. We stress that *any* filter bank

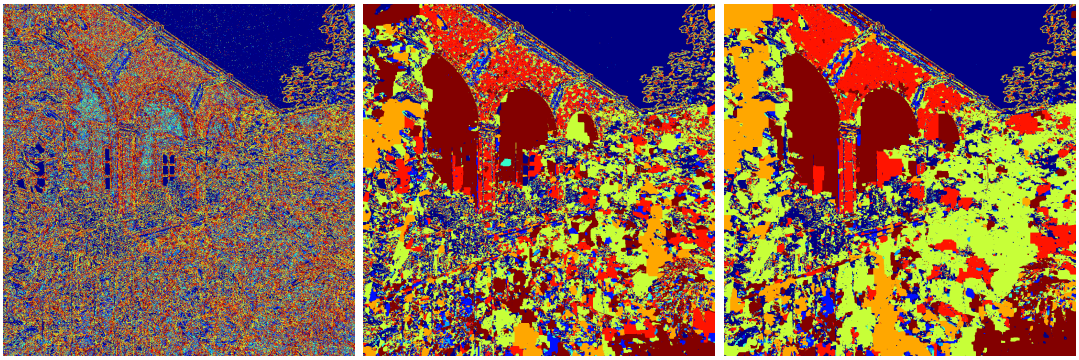


Figure 5.2: Filter selection maps with increasing coherency for the SANMIGUEL scene for a filter bank of 8 filters. The MSE reduction of the coherent selections is only slightly less compared to the optimal selection.

could be used and refer the interested reader to the according publications for more details. Still, we assume a set of reasonable filters and settings, which means that for most pixels a choice exists, which represents an improvement over the initial MC estimate.

To reach our goal, we estimate the reconstruction error of each entry in $\mathbb{F}$ at a small number of pixels (Sect. 5.1.1) and show how to optimize their locations (Sect. 5.1.2). The sparse estimates are interpolated based on a smoothness assumption (Sect. 5.1.3) to derive per-pixel error estimations. These will then be used as input to a labeling process to choose the optimal entry in $\mathbb{F}$ per pixel. The latter is solved via a graph-cut approach (Sect. 5.1.4) to avoid visual artifacts due to inconsistent or inappropriate filter choices. An overview of our algorithm is given in Fig. 5.3.

Figure 5.3: Overview of our proposed framework.

5.1.1 Filter Caches

Our approach is inspired by ir/-radiance caching algorithms where the incident indirect lighting is computed for a small subset of pixels (the caches) and later interpolated. In

this spirit, we compute a high-quality radiance estimate for a small subset of pixels. To obtain these so-called *filter caches*, more samples are computed and because the MC error initially decreases quickly [91], even a slightly elevated number of samples leads to a significant improvement of the incoming radiance estimate. In consequence, it is possible to obtain a good error estimate Err at a pixel cache location p with value $\mathbf{C}(p)$ for any $\mathbf{F}_i \in \mathbb{F}$

$$Err(p, \mathbf{F}_i) \approx \|\mathbf{F}_i(p) - \mathbf{C}(p)\| .$$

To roughly maintain the overall rendering cost, we create two sample sets out of the sample budget based on user-defined parameters, one part used for uniformly sampling the image plane to create the filter input $\mathbf{N}$ and the rest to compute the cache entries. Given the initial per-pixel sample budget b, a cache sparsity $s \in [0, 1]$ (a value of 0 places a cache at each pixel and a value of 1 results in no caches at all), and the number of samples per pixel n used to compute $\mathbf{N}$, the additional samples per cache c are given by $\frac{b-sn}{1-s}$. To acquire a robust radiance estimate close to the reference, as we treat $\mathbf{C}(p)$ as ground truth, the sparsity has to be high; we used between 0.85 and 0.98 of sparsity in our test scenes.

5.1.2 Filter Cache Placement

As our intermediate goal is to interpolate the error values between the caches for each filter, a good placement is crucial. For an even spread, we can distribute their location according to a blue-noise power spectrum [23]. Nonetheless, to better capture error variations, more caches should be placed in regions with varying error, which are unknown.

Instead, we base our adaptive cache placement strategy on three insights. First, the variance within $\mathbb{F}$ for a pixel p indicates how crucial a good filter selection is. If the variance is high, a wrong filter choice will introduce a large error. Vice versa, if all $\mathbf{F}_i$ are the same, the choice is unimportant. Second, pixels with already low variance in the MC estimator for $\mathbf{N}$ are likely to provide a more robust radiance estimate with less residual noise for the cache's sampling rate. Third, the overall image domain Ω should be roughly covered with a maximum distance between the caches.

Following the first two insights, we compute a joint probability distribution function (PDF) $\mathbf{P}_{\mathbb{F}\mathbf{N}}$ to drive the cache positioning:

$$\mathbf{P}_{\mathbb{F}\mathbf{N}}(p) = \frac{\mathbf{P}_{\mathbb{F}}(p) \cdot \mathbf{P}_{\mathbf{N}}(p)}{\sum_{p \in \Omega} \mathbf{P}_{\mathbb{F}}(p) \cdot \mathbf{P}_{\mathbf{N}}(p)}, \tag{5.1}$$

where $\mathbf{P}_{\mathbb{F}}$ and $\mathbf{P}_{\mathbf{N}}$ are the PDFs for importance sampling based on the filter bank variance and the MC variance of $\mathbf{N}$. We use the per-pixel filter bank variance directly as PDF and set $\mathbf{P}_{\mathbb{F}}(p) := \mathbf{F}_{\sigma^2}(p) = \frac{1}{m} \sum_{i=1}^{m} \left(\mathbf{F}_m(p) - \mathbf{F}_{\mu}(p)\right)^2$, where $\mathbf{F}_{\mu}$ is the per-pixel mean of the filter bank. For $\mathbf{P}_{\mathbf{N}}$, we use a Gaussian probability model and define

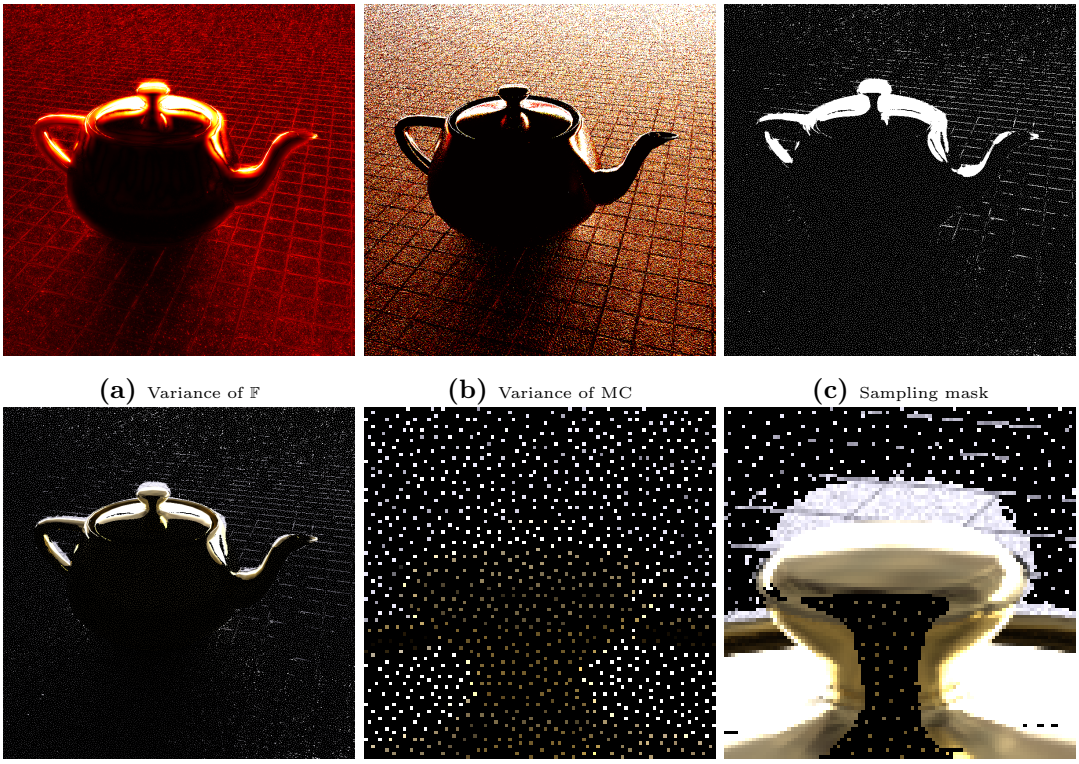
$$\mathbf{P}_{\mathbf{N}}(p) = \frac{1}{\sigma_r \sqrt{2\pi}} e^{-\frac{\mathbf{N}_{\sigma^2}(p)}{2\sigma_r^2}},$$

where $\mathbf{N}_{\sigma^2}$ is the variance of the MC estimator. Although $\mathbf{N}_{\sigma^2}$ is unknown, it can be approximated by the empirical sample variance for each pixel using n samples (cf. [69, 98]). Here, σ_r is a global, user-defined parameter, which we set to 0.15 for all our scenes.

Directly sampling the total number of caches m_{total} via $\mathbf{P}_{\mathbb{F}\mathbf{N}}$ leads to cluttered cache locations and potentially larger image regions without caches, which leads us to the third insight. We use our previous importance sampling strategy to produce $m_{\text{importance}} := \kappa \cdot m_{\text{total}}$ caches and $m_{\text{poisson}} := m_{\text{total}} - m_{\text{importance}}$ caches using standard Poisson sampling. Here, $\kappa \in [0, 1]$ is a user-defined number, which balances between both strategies.

To avoid duplicate caches, we first draw m_{poisson} samples from the Poisson distribution and remove the sampled pixels from the PDF computation of the importance sampling given by Eq. (5.1). Afterwards, we draw the $m_{\text{importance}}$ samples from the resulting PDF using 2D importance sampling [91]. As the parameter κ is set only once, the Poisson distribution can be precomputed. A similar sampling could be achieved with a pure variable density-based Poisson Distribution, e.g. [49], but these are often costly to compute. Our approach is cheap as the uniform Poisson samples can be precomputed.

An example of our approach is given in Fig. 5.4; the samples of our importance sampling approach visibly gather around object boundaries and high frequency edges, as these are often difficult to reconstruct for many filters.



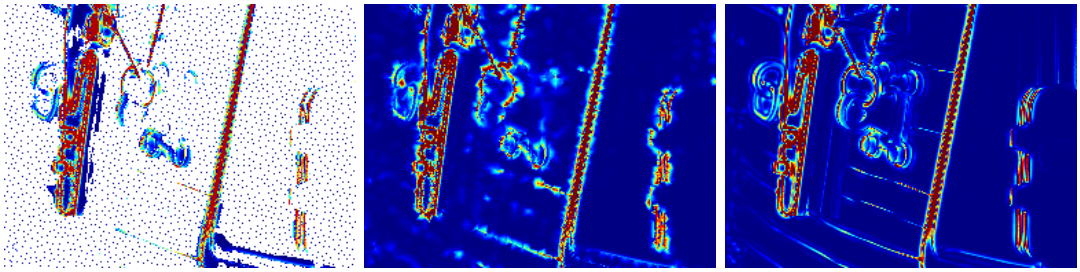
(a) Variance of $\mathbb{F}$ **(b)** Variance of MC **(c)** Sampling mask

(d) Representative pixels **(e)** Poisson sampling **(f)** Poisson + Importance

Figure 5.4: Adaptive cache placement. (a) Magnitude of the variance of the filter bank, (b) Magnitude of the variance of the MC estimator, (c) Binary mask of the representative pixels using our importance sampling approach, and (d) the representative radiance values. (e) Close-up with standard Poisson sampling and (f) with our proposed combination of Poisson and importance sampling.

5.1.3 Dense error reconstruction

Given the error estimation at the cache locations for each $\mathbf{F}_i \in \mathbb{F}$, we want to estimate a dense error for all remaining pixels, which will be the input to our filter selection approach. We compared several reconstruction techniques of sparsely sampled images, including PDE-based inpainting [8], Total-Variation inpainting [13], and ACT [36]. An excellent comparison study for several image-based sparse-reconstruction techniques has been presented in [105]. The study shows that Compressed Sensing and Delaunay triangulation are the best choices for sparsely-distributed samples with a high degree of sparsity. As we have to perform multiple interpolations for each $\mathbf{F}_i$, we chose Delaunay triangulation because it provides a good trade-off between quality and performance.



(a) Sparse squared error **(b)** Inpainted squared error **(c)** Reference squared error

Figure 5.5: From a sparse estimation of the squared error, we use an inpainting based on Delaunay triangulation to compute a dense estimation, which resembles the reference squared error. Note that the uniform areas are sampled with fewer caches by our adaptive cache placement.

Further, once computed, it can be re-used for each filter in the bank.

We use the square of the L2-norm as error distance metric (Fig. 5.5) and interpolate the error of each reconstruction technique $\mathbf{F}_i$ between the caches to create a dense error map $\mathbf{D}_i$ for each filter.

5.1.4 Filter Composite

Given the dense error estimate $\mathbf{D}_i$, a straightforward solution to minimize the MSE of the final result would be to select the filter $\mathbf{F}_i$ with the lowest estimated error per pixel, but this may lead to visual artifacts in form of seams. Instead, we write the problem of selecting the optimal filter per-pixel as a multi-labeling optimization problem. An optimal labeling $\mathbf{L} : \Omega \to 1 \dots m$ can be found by minimizing the energy

$$E(\mathbf{L}) := E_{\text{Data}}(\mathbf{L}) + \lambda \cdot E_{\text{Smoothness}}(\mathbf{L}), \tag{5.2}$$

λ allows us to balance between data and smoothness terms, which are defined as follows;

$$E_{\text{Data}}(\mathbf{L}) := \sum_{p \in \Omega} \mathbf{D}_{\mathbf{L}(p)}(p) \tag{5.3}$$

$$E_{\text{Smoothness}}(\mathbf{L}) := \sum_{\{p,q\} \in \mathcal{N}} V(p, q, \mathbf{L}(p), \mathbf{L}(q)), \tag{5.4}$$

where $\mathcal{N}$ is the set of interacting pairs of pixels and $V(p, q, \mathbf{L}(p), \mathbf{L}(q))$ is a label cost function. We follow Agarwala et al. [1] and define this cost function to match color

and gradients of neighboring pixels:

$$V\left(p, q, \mathbf{L}(p), \mathbf{L}(q)\right) = X + Y$$

with

$$X = \left\|\mathbf{F}_{\mathbf{L}(p)}(p) - \mathbf{F}_{\mathbf{L}(q)}(p)\right\| + \left\|\mathbf{F}_{\mathbf{L}(p)}(q) - \mathbf{F}_{\mathbf{L}(q)}(q)\right\|$$
$$Y = \left\|\nabla\mathbf{F}_{\mathbf{L}(p)}(p) - \nabla\mathbf{F}_{\mathbf{L}(q)}(p)\right\| + \left\|\nabla\mathbf{F}_{\mathbf{L}(p)}(q) - \nabla\mathbf{F}_{\mathbf{L}(q)}(q)\right\|$$

where $\nabla\mathbf{F}_i(p)$ is the (horizontal and vertical) gradient of the filtered image $\mathbf{F}_i$ at p. Eq. (5.2) can be solved efficiently within a known factor of the global minimum using a graph-cut optimization [11].

Fig. 5.6 shows the effect of the global optimization. Local per-pixel filter selection leads to small erroneous patches where neighboring labels differ and creates visually-disturbing artifacts, which are robustly removed by our graph-cut approach.

Gradient-Domain Fusion In image-stitching applications [1, 111], it has become common practice to add a final Poisson integration step to adjust colors along the seams of neighboring regions. For very low sampling rates, it also smooths out juxtaposed filter regions. We use two Jacobi iterations of the Poisson solver to smooth the most visible seams without affecting the overall MSE.

Additionally, the Poisson formulation can be used to enforce the filter-cache radiance as a constraint for the image reconstruction. In practice, for the low sample count that we target, the remaining variance in the caches is usually similar to the reconstruction. Consequently, integrating the caches proved counter-productive.

5.2 Results

We implemented larger parts of our method in MATLAB R2014b without the use of multi-threading. For the multi-label graph-cut solver, we use the algorithm proposed in [11] and implemented it in NVIDIA's CUDA 6.5 on top of the binary graph-cut implementation provided by the CUDA NPP library. We also implemented the joint-bilateral filter with the modified distance function from [69] using CUDA. For BM3D denoising [19] and the modified non-local means filter by Rousselle et al. [100], we

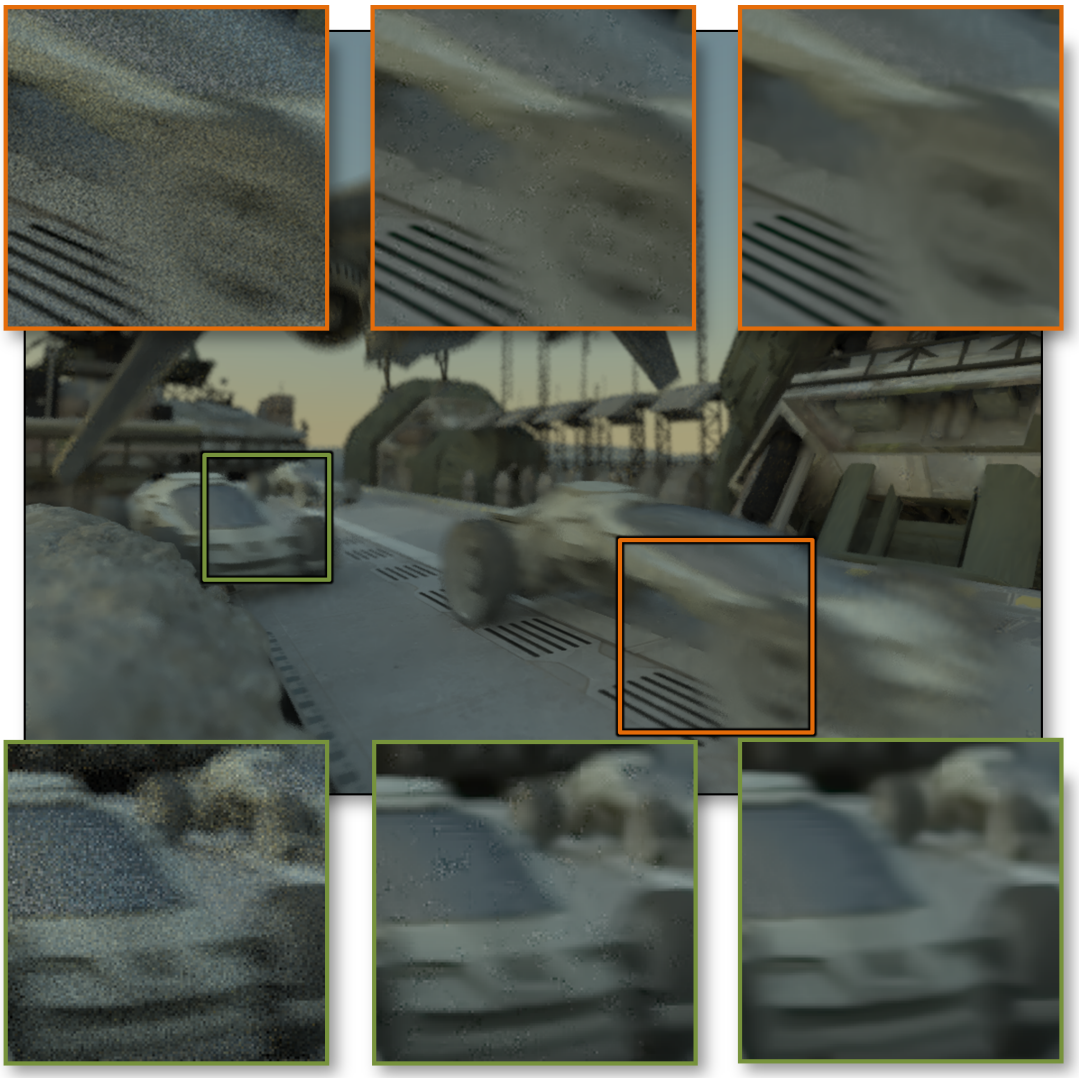


Figure 5.6: Noisy image with 32 spp (left inset), the result using only an interpolated local error estimation (middle inset), and the graph-cut version (right inset) of the SCIFI scene.

used the original implementations. All statistics were measured on an Intel Core i7-2600, 3.40 GHz and 16 GB RAM PC with an NVIDIA GeForce 780 GTX running on Windows 7, 64-bit.

The test scenes and the input and reference data were created using the PBRT2 system [91]. We used nine test scenes for our evaluations and comparisons (resolution in pixels is given in brackets) - Sanmiguel(1024x1024), Sibenik (1024x768), Teapot (800x800), Toasters (512x512), Chess (750x1000), Poolball (1024x1024), Dragon (1024x1024), Conference (1024x1024) and Scifi (1024x768). The scenes cover a variety of MC effects including global illumination, depth-of-field, motion blur, glossy materials and participating media. All reference solutions have been computed with 20000 spp.

5.2.1 Parameter and Error Evaluation

In the following, we evaluate the influence of the different parameters of our approach to derive an optimal setting used in all the following results. Additionally, we investigate the error of our method compared to the optimal filter selection. For all experiments, we set a total sample budget of 32 samples times the number of pixels. The samples used for $\mathbf{N}$ are uniformly distributed among the image in our approach.

Sampling and Sparsity We start by evaluating the influence of the number and quality of filter caches. To find the optimal sample distribution, we vary the number of samples used for $\mathbf{N}$, as well as the number of filter caches. The results for the Sibenik scene are shown in Fig. 5.7. The trade-off between sparsity s and cache sampling rate c has in general a lower impact on the MSE in comparison to varying the number of samples for $\mathbf{N}$. This shows that the reduction of the interpolation error from lower sparsity is outweighed by the increase in the variance of the cache radiance estimates. Optimal parameters are achieved between three-fourths and seven-eighth samples assigned to $\mathbf{N}$ and a sparsity of approximately 95%, which results in a mean distance of roughly four pixels between the caches.

Approximation Error & Adaptive Cache Placement We evaluate the influence of adaptive cache placement vs. a blue-noise distribution for several test scenes. To this extent, we vary the parameter κ to interpolate between the two extremes as described

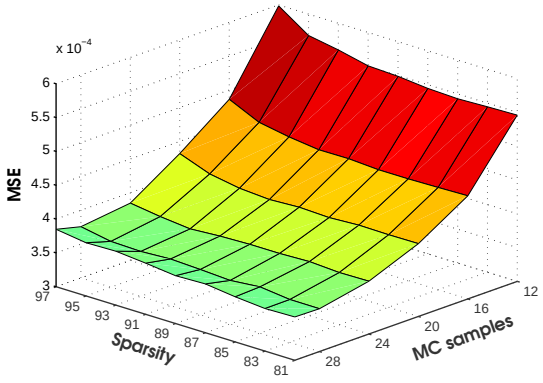


Figure 5.7: Parameter evaluation. While the overall sampling budget is fixed, the sampling rate for the noisy input image and the number of filter caches is varied. Optimal parameters are achieved with between three-fourths and seven-eighth samples assigned to the noisy image and a sparsity of approximately 95%, which results in a mean distance of roughly four pixels between the caches.

in Sect. 5.1.2. In addition, we are interested in the significance of the two possible error sources of our approach; interpolation errors and errors introduced by residual variance in the cache radiance values. To gain further insights into each of them, we also measure the MSE when using ground truth radiance at the caches instead of the variant estimates $\mathbf{C}(p)$.

Adaptive placement consistently outperforms uniform placement and our mixed importance sampling decreases the overall MSE down to 90% for the SIBENIK scene, 84% for the TOASTERS scene, 63% for SANMIGUEL, 86% for the TEAPOT scene and even down to 43% for the DRAGON scene. We used values of $0.5 - 0.7$ for κ in our scenes.

As expected, interpolation from sparse caches is our main source of error in most scenes (72% on average) when compared to an optimal filter selection (Fig. 5.8), while error from residual noise in the caches is comparably small (28% on average). An exception is the DRAGON scene where interpolation works exceedingly well due to large homogeneous image regions and residual cache variance contributes more strongly to the overall error (80% on average).

Regularization We varied the smoothness parameter λ, controlling the influence of the smoothness term in the graph-cut labeling Eq. (5.2) and evaluated the MSE. The

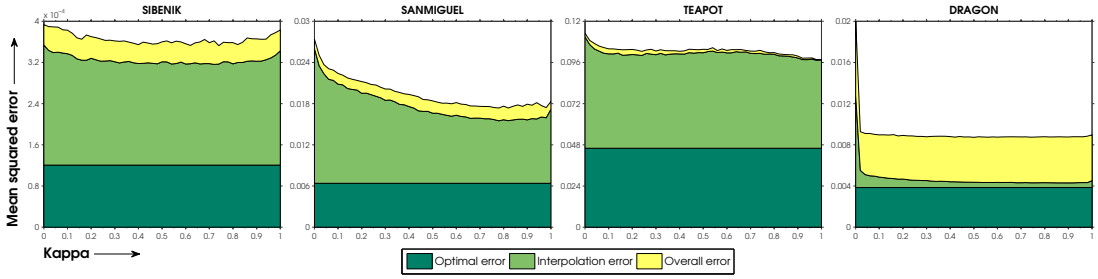


Figure 5.8: Influence of parameter κ on the MSE for the adaptive cache sampling strategy using 16 spp for **N**, 95% sparsity, resulting in 176 spp for each cache entry, and the filter bank from Sect. 5.1. The error from an optimal filter selection (dark green) is compared to the error due to interpolation (using ground truth radiance at the caches, shown in green) and the overall error (including interpolation and residual noise in the caches, shown in yellow). The mixed importance sampling decreases the overall MSE for all test scenes: Sibenik (90%), Sanmiguel (63%), Teapot (86%), Dragon (43%).

regularized version decreases the overall error compared to the non-regularized version ($\lambda = 0$) for all scenes up to 27% (Fig. 5.9). Hereby, small filter patches in the resulting images are removed, which otherwise could appear as visible artifacts (Fig. 5.6). The graph cut operates on the color values of $\mathbb{F}$, hence, the optimal choice of λ depends on the dynamic range of the scene radiance. When λ is chosen too large, the error is increased again due to over-smoothing of the final labeling.

5.2.2 Timings

We evaluate the runtime of our method in Table 5.10 using 32 spp and 90% sparsity. It can be seen that our method accounts for only a small portion of the total time: 3% - 6% for the Sibenik scene, 6% - 9% for Conference, 3% - 6% for Sanmiguel, 8% - 12% for Teapot and 4% - 7% for the Dragon scene. Most time is consumed by the rendering process and the filter-bank creation. Our method's most costly aspect is the graph-cut solver, which can be seen in the increase of the compositing time between 4 and 8 filters, due to the quadratic complexity in terms of labels.

5.2.3 Comparisons

We compared our technique to a variety of state-of-the-art MC denoising techniques. For all comparisons, we used the original source codes kindly provided by the respective

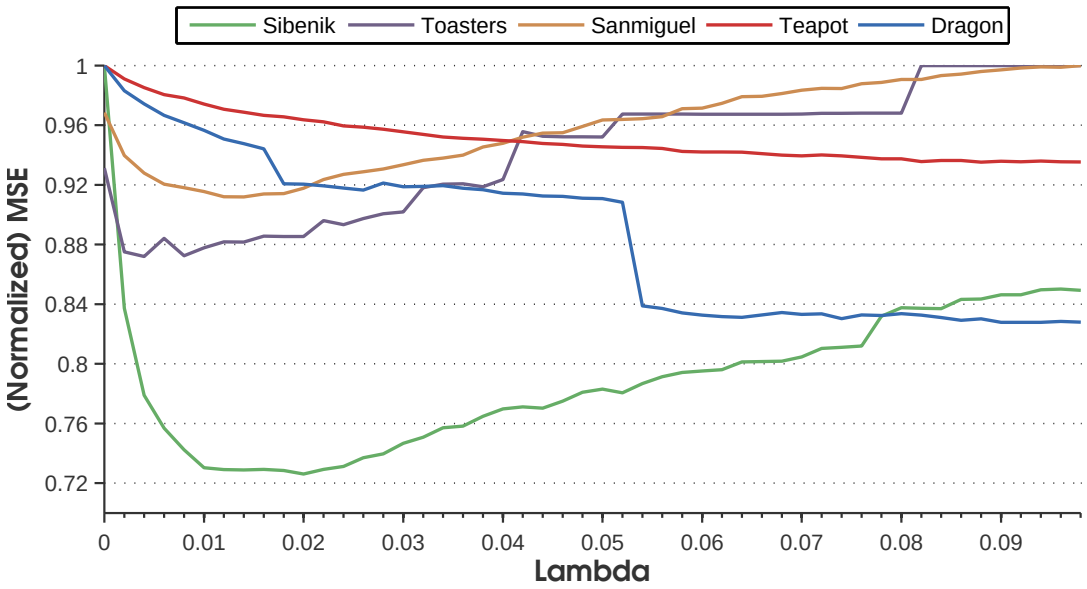


Figure 5.9: Influence of the parameter λ for the graph-cut filter selection (using the parameters from Fig. 5.8). In comparison to un-regularized selection, choosing the *optimal* λ reduces overall MSE for Sibenik (73%), Toasters (87%), Sanmiguel (91%), Teapot (94%), and Dragon (83%).

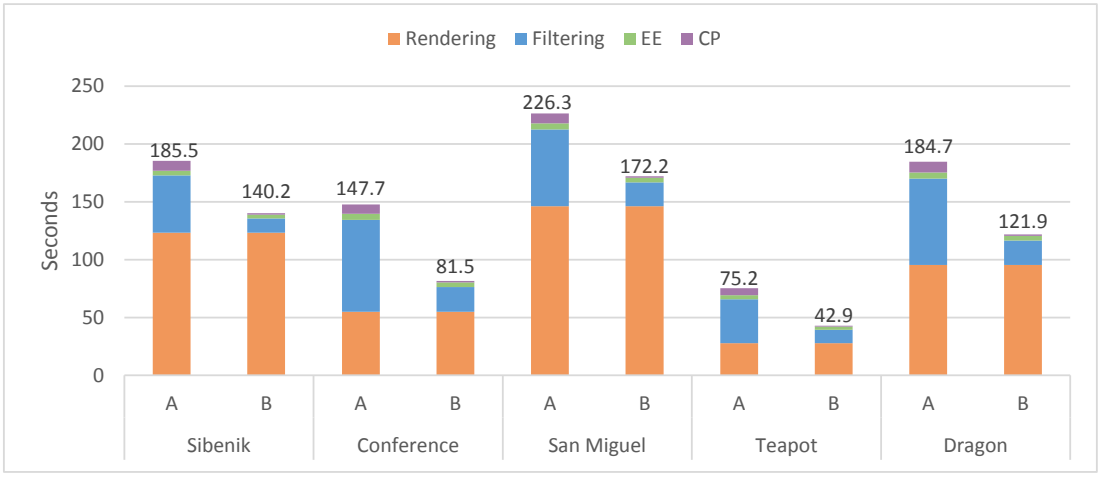


Figure 5.10: Timings for rendering (with 32 spp), filter bank creation (A uses 8 joint-bilateral filters, B uses 4 BM3D filters), error estimation (EE) including cache location sampling and dense interpolation, and compositing (CP) including graph cut and Poisson integration. All timings are in seconds.

authors. All parameters of these techniques were set to values proposed in the respective publications and the same overall sample budget was used for a fair comparison.

SURE To evaluate the quality of our filter selection, we compare our technique to SURE-based filter selection which was proposed first for MC denoising in [69]. Because SURE only works for differentiable filters, we use a filter set consisting of four joint-bilateral filters (using similar parameters settings as [69]) for both SURE and our approach for a fair comparison. We use 32 spp for the SURE method to create the filter bank, while our method uses only 28 spp for the noisy estimate $\mathbf{N}$, from which the filter bank is constructed. Both noisy images are created with uniform sampling. We distribute the rest of the samples to the cache pixels, which we create with 95% sparsity, i.e. each cache is computed from 108 samples. Results for the Sibenik and Conference scenes are shown in Fig. 5.11.

For both scenes our technique has fewer visible artifacts and an overall MSE reduction of up to 54% for the Sibenik scene and 64% for Conference compared to the SURE-based selection. Even when choosing the best filter only locally without the graph-cut and Poisson-integration step, we still achieve an improvement in MSE by 43% (Sibenik) and 47% (Conference). Potentially, our results could be improved even further by using higher quality filters, including non-differentiable ones, in $\mathbb{F}$.

GID We compare our method to the General Image Denoising (GID) framework [50], a framework for adaptive filtering and variance estimation (Fig. 5.12). GID uses a wavelet-based noise metric to estimate the standard deviation of the noise per pixel and then selects an optimal filter from a series of high-quality filters (BM3D or BLS-GSM). The BM3D and BLS-GSM are only applicable to low dynamic range images, which is why images have been tone mapped beforehand using a gamma correction. The test sequences are the Toasters, Chess and Poolball scene using BM3D filters and the parameters suggested by the authors. For our method, we create a filter set consisting of four BM3D filters with uniformly distributed parameters, while GID even optimized the filter parameters for each scene. Additionally, GID performs adaptive sampling. We tested configurations with 8, 16 and 32 spp on average. For all 3 scenes, we used a sparsity of 95%.

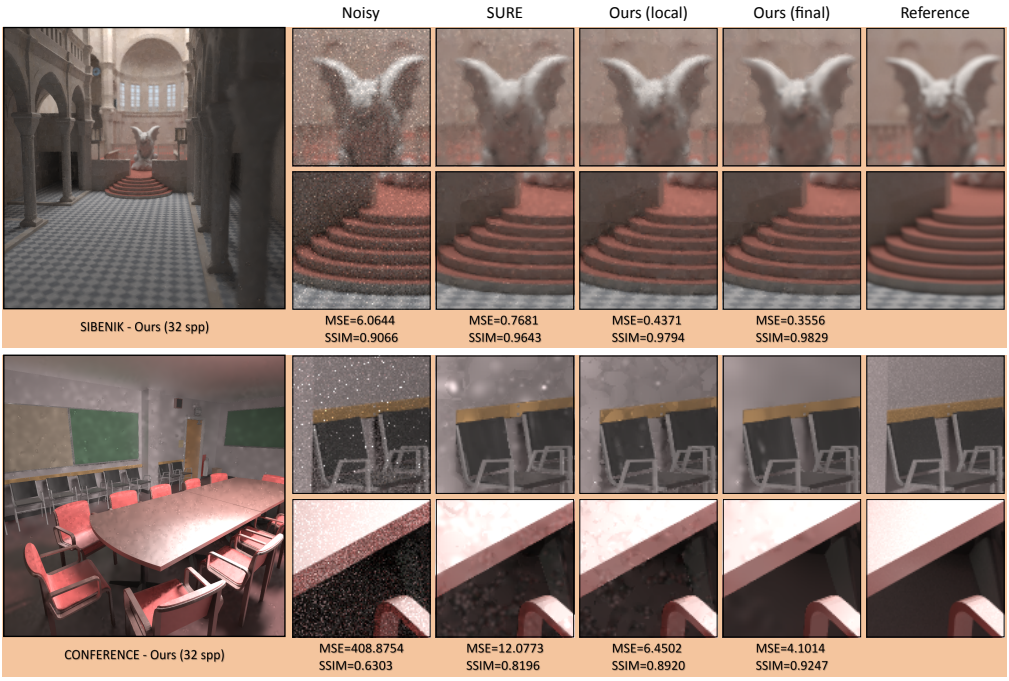


Figure 5.11: Comparison with SURE [69]. Top: Reconstructed result and associated filter selection map. All reconstruction results, except for the reference solution, use the same sample budget. MSE is scaled by a factor of 10^3.

We achieve an MSE reduction of 13% - 64% (51% on average). The GID approach has problems distinguishing between noise and high frequencies in the image signal and smoothes over them. This shortcoming is visible in areas with high-frequency textures (e.g. in the chess scene, second row).

Similar to previous approaches, the GID estimator suffers from variance for low sampling rates which forces the approach to heuristically smooth the noise map before filter selection. The framework supports arbitrary image-denoising filters, however, they only estimate the variance of the noise and not the MSE itself. Therefore, only results for one family of filters at a time have been shown. The GID algorithm has no means to compare the reconstruction quality of different filters directly. Our approach differs in this sense, as we estimate the MSE directly per filter and are able to compare arbitrary reconstruction results.

RD In Fig. 5.13, we compare our method with the Robust Denoising (RD) framework [100], which is the currently best-performing reconstruction technique of all tested approaches. RD uses three specialized non-local means filters with different sensitivity to the image colors. Additionally, it uses a tailor-made filter selection algorithm for these three filters based on SURE. We use the same three filters for our filter bank.

The results shown in Fig. 5.13 for both approaches are quite similar. For 16 spp (32 spp), our technique shows a MSE reduction of up to 25% (16%) for the Dragon scene compared to RD. On the Teapot scene, our approach performs slightly worse and the MSE increases by 14% (13%). For the Conference scene, both approaches yield similar results, with an MSE decrease by 9% (7%) using our method. A possible explanation could be that a sparse cache sampling potentially misses peaks in the highly glossy material of the teapot, resulting in worse error estimates in the dense error maps. However, it should be noted that our approach is a general framework for arbitrary reconstruction techniques, whereas RD is a specifically customized approach. Our images show slightly more noise compared to the results presented in [100] as we omit their final filtering step for a more direct comparison.

Though not tested yet, we could potentially use the final results from the SURE, GID and RD frameworks for our input and locally choose an optimal one, which illustrates the flexibility of our approach.

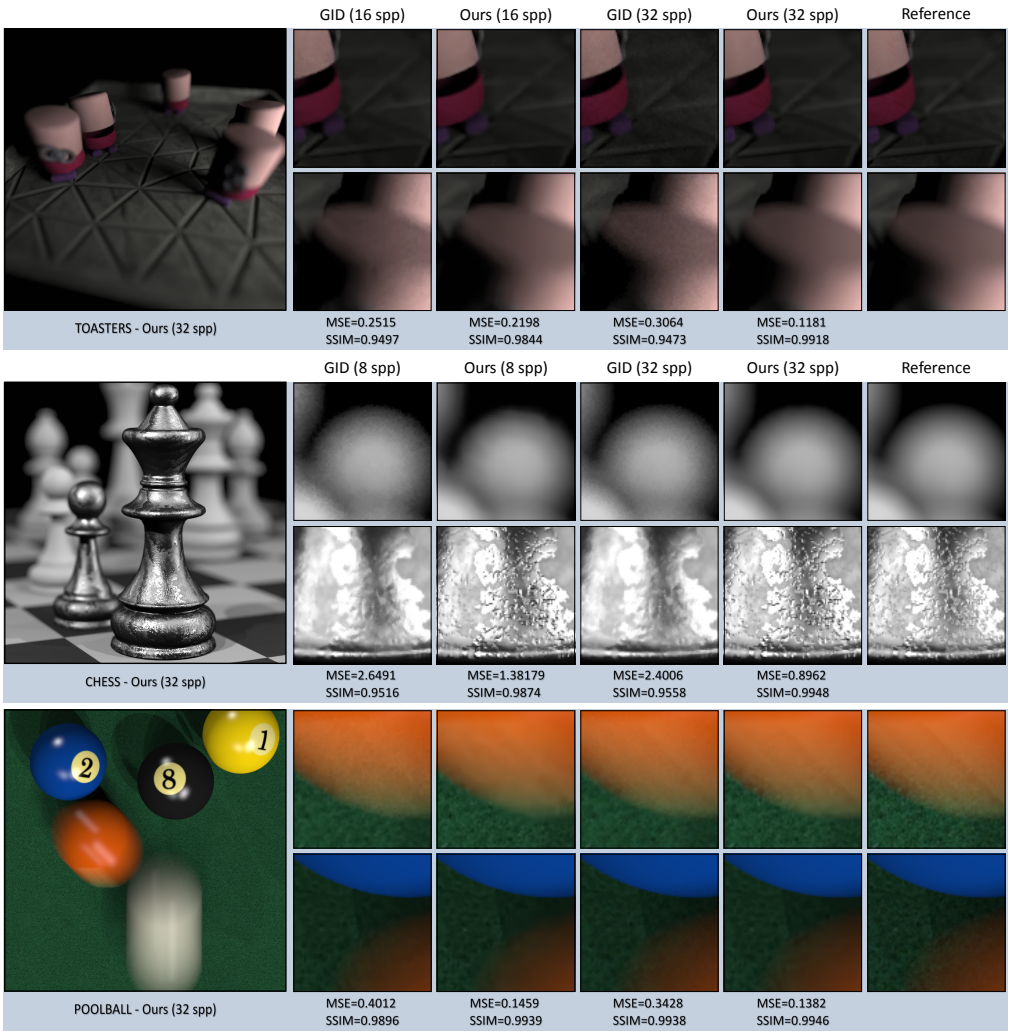


Figure 5.12: Comparison with the General Image Denoising (GID) framework [50] for 8, 16, and 32 spp, on average. Our approach uses four different BM3D filters. The MSE is scaled by a factor of 10^3.

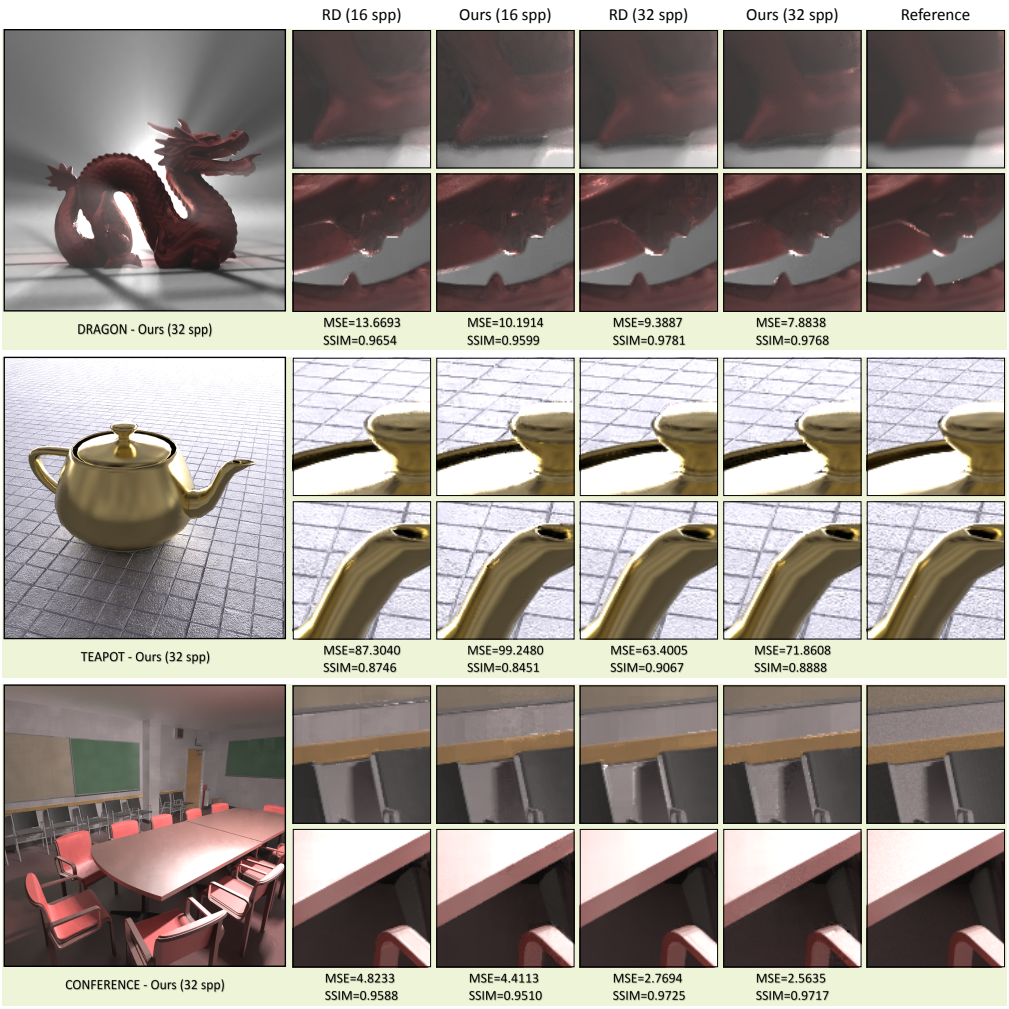


Figure 5.13: Comparison to the Robust Denoising (RD) technique [100] for 16 and 32 spp, on average. For all scenes three customized non-local means filters are used. MSE is scaled by a factor of 10^3.

5.3 Discussion

Based on our experiments from the previous section, using filter caches appears to be a fruitful research direction to develop more robust error estimators as our solution outperforms many previous approaches. It also shows that sample count alone can be much less crucial than appropriate settings for the reconstruction method. This finding indicates that solutions like ours, which opt at making better filter choices, have great potential for improving image reconstruction in the future.

A limiting factor can be residual noise in the filter caches, or an inferior error interpolation, which can both affect the quality of our filter selection. Both problems could potentially be tackled by compressed-sensing [105] in combination with cross-validation [10], but we currently opted for a computationally more efficient solution.

In the current version, our adaptive placement of caches tends to avoid selecting outliers stemming from the MC rendering process as caches, which are, unfortunately, even preserved by some filters (e.g., BM3D or BLS-GSM). This situation violates our assumption of local error smoothness, although it is more a filter limitation. Further, preprocessing the individually-filtered images using a spike-noise reduction should alleviate these problems.

Adaptive MC sampling is often based on intermediate reconstruction results to steer the distribution of samples. Our approach offers many degrees of freedom as samples can be added to the caches, be integrated in our noisy estimate $\mathbf{N}$, or can even be used to create new caches. The gain could be high, but such measures make the problem also substantially more complex.

This concludes Part II of the thesis. In this part, we presented an interactive filtering method for removal of noise from global illumination and depth-of-field, as well as a robust and generic framework for error estimation applicable to adaptive reconstruction. These approaches can be used to accelerate a variety of applications – from interactive settings to production rendering – which are based on MC rendering. In the next part of the thesis, we will address another challenge: memory consumption of acceleration structures.

Part III

Memoryless Acceleration

6

Background

6.1 Motivation

Acceleration structures (AS) are essential for speeding up ray tracing and constitute a major area of research for many decades. Driven by consumer demands, scene complexity is ever-increasing to constantly achieve higher degrees of realism and suitable AS are needed to make it possible. The quality of an AS is primarily measured by three characteristics: traversal time, construction time, and memory footprint. The ultimate goal is to reduce the time spent on intersecting rays with the scene and hence, traversal time is an obvious quality criterion. When a large amount of rays is traced, the total rendering time is dominated by ray traversal and the setup phase, i.e. the construction, of an AS consumes merely a negligible amount of time. However, with the rise of interactive ray tracing for dynamic scenes [86, 121] this ratio has been shifted and construction timings gained more and more interest. Today's research focus on the total time-to-image including both the construction *and* the traversal time.

The need for AS in ray tracing and the accompanying memory requirements were always considered a disadvantage of ray tracing when compared to scan-line and rasterization techniques. Since the advent of many-core ray tracing (e.g. on GPUs), memory footprint becomes a renewed concern as performance on these architectures is often dominated by memory throughput. Also, with increased scene detail and complexity, available on-board memory resources can become a bottleneck. If a model and its AS do not fit into main memory, slow disk I/O performance dominates rendering time [123, 131]. Savings can spare the need for out-of-core rendering and make memory

available for more geometry, textures, etc. Therefore, reducing the memory footprint of AS while maintaining high traversal and construction times is an interesting and important challenge in current research.

In the following chapter, we will present a new AS concept that originates from the BVH, which can be considered the current state-of-the-art for ray tracing. We introduce a novel implicit representation of a complete object space partitioning (OSP) that requires no memory at all. Although completely memoryless, our approach allows for interactive construction and traversal performance and is perfectly suited for many-core architectures.

6.2 Related Work

Over the years, efforts have been made to minimize the memory usage of the classic BVH from Kay and Kajiya [52] which consists of six bounding planes per node perpendicular to the world coordinate axes. One common way to reduce the number of nodes of a BVH is to use a higher branching factor [20, 31], but this often comes at the cost of reduced performance.

Bounding Planes Other more efficient approaches merge or store only a subset of the bounding planes and can be seen as derivatives of the classic BVH. Several authors proposed to remove half of the bounding planes due to the observation that the twelve planes of the children of a node always share six sides with their parent [32, 33, 51]. By saving the active ray interval, a hit or miss can be conservatively estimated with even less planes and hybrid techniques have been thoroughly investigated during the last years which try to combine the benefits of kd-trees and BVHs. These *hybrid* BVHs were developed independently by several researchers [28, 44, 116, 129, 132, 134]. Zachmann et al. [132] proposed a single bounding plane approach for collision detection with oriented bounding boxes. A similar representation but with axis-aligned bounding planes and a fast global construction heuristic was used by Wächter and Keller [116]. Woop et al. [129] showed a hardware implementation of a similar structure which uses two opposing bounding planes per node, called B-KD tree. The DE-Tree by Zuniga and Uhlmann [134] shows similarities with the B-KD tree but uses wide object isolation to keep larger objects at earlier levels of the hierarchy plus a higher branching factor.

Havran et al. [44] adapted a version of the SKD-tree by Ooi et al. [81] and extended them to incorporate different node types in order to improve efficiency. Eisemann et al. [28] propose to use a single, axis-oriented slab per node. These sparse bounding slab representations may reduce the memory requirements up to a third, compared to the classic BVH.

Reducing Precision Another direction for memory reduction is to lower the precision of the bounding planes which are stored using six full-precision floating point values in standard BVHs. This is valid, as long as the hull volume still encapsulated all the primitives inside – it is not forbidden for the bounding volume to be larger as needed, although it can lead to false positives and reduced traversal performance. Mahovsky and Wyvill [71] investigated a hierarchical scheme for encoding bounding volumes in a BVH relative to their parent node that reduces the storage requirements by 63%-75%. Cline *et al.* [16] used a hierarchical encoding scheme, compressing a node to 12 bytes, plus a high branching factor of four and implicit encoding of the child pointer due to a heap like data structure. Segovia and Ernst [104] follow Mahovsky's approach and use 4 bytes per BVH node, but extend it in two important ways. Despite compressing the triangle data as well, they store the BVH nodes in clusters to reduce the size of all references to child nodes and they use a two-level BVH which uses both uncompressed BVH nodes for the top levels and compressed nodes for the rest of the hierarchy. This idea of a two-level hierarchy for compression was first presented in [64] and we will make use it in our approach to allow a convenient trade-off between memory reduction and performance. Kim et al. [56] also build upon the two-level approach and additionally introduce tree templates to reduce the number of necessary child pointers. Wächter and Keller [118] proposed a new termination criterion for spatial subdivision schemes and use a fixed memory footprint, but rendering efficiency quickly deteriorates if less than five bytes per scene primitive are used. Kim *et al.* [57] apply a compression algorithm to reduce the memory footprint of a BVH. A drawback of their method is that they introduce a decompression cost during the traversal leading to a performance decrease. However, for large scenes (especially ones which cannot be rendered without out-of-core techniques) this performance loss is cancelled by the possibility to keep the whole BVH in the main memory and improved cache usage. Their approach achieves a compression ratio of 12:1.

Divide-And-Conquer Ray Tracing A recent way of removing the need for an acceleration structure is *divide-and-conquer ray tracing* which was proposed for object subdivision by Keller and Wächter [55] and for spatial subdivision by Mora et al. [79]. To our best knowledge, this is the only alternative approach which implicitly models an acceleration structure and – although a very different approach – can be considered a direct competitor to our method in Chap. 7. At a closer look, divide-and-conquer ray tracing is more a general concept than an actual acceleration structure. In each traversal step, the method first compute the bounding box for the current primitives of the object. In the next step, the active rays that intersect the box are computed. The primitives are partitioned into two sets according to a chosen splitting plane. The algorithm is then recursively called for the active rays and the new partition. If the number of primitives is below a certain threshold the active rays are directly tested for intersection. If only primary rays are traced the algorithm have an almost perfect time to image as only those parts of the hierarchy are created which are actually traversed. Occluded parts are left not partitioned. A drawback of the method is that the implicit reconstruction has to be repeated for each ray batch which can possibly result in a non-negligible computational overhead for complex lighting simulations with many shadow and secondary rays. Also, according to Mora [79] an efficient parallel GPU implementation poses difficulties and has not yet been further investigated.

Our contribution **Geometry Presorting for Implicit Object Space Partitioning** (IOSP) in Chap. 7 is a fully implicit AS without any additional memory requirements. We built upon several modifications to the standard BVH and reduce the number of information that is stored for each BV, use only a subset of the six bounding planes, and remove child and primitive pointers by implicit indexing. A key characteristic of our approach is that we derive the position of the bounding planes directly from the contained geometry of each node instead of saving it explicitly. Our approach is inspired by multidimensional nearest neighbor search structures [103], where primitive references in inner nodes can be used for early pruning of sub-trees. This novel concept nicely benefits from the parallelism and computational power of current GPUs both in construction as well as rendering. This work was presented at the *Eurographics Symposium on Rendering (EGSR)* in 2012 and published in the accompanying proceedings.

7

Geometry Presorting for Implicit Object Space Partitioning

To remove any additional memory of an acceleration structure, we propose an implicit representation of a complete object space partitioning (OSP) in this chapter. The core idea is to presort the geometry, access the portion that spans each node directly and reconstruct the bounding planes on the fly. In each traversal step, we load the primitives bounding the node and reconstruct the slabs on the fly. This step reduces memory requirements to only four bytes per node, leaving only the child/geometry pointer. It also reduces the number of nodes in the hierarchy as triangles are tested in all nodes and not only in leaf nodes. Finally, we can remove even these last four bytes by representing the hierarchy as a heap, resulting in an OSP that requires no memory at all: it is represented completely implicit by triangle order. Our approach is easy to parallelize and well suited for many-core processors. A major advantage of our approach compared to divide-and-conquer ray tracing is that a resorting is only necessary if the geometry changes, no rebuilt is required if only the viewpoint or lights are moved. We additionally present a parallel construction technique, demonstrating that our approach is applicable to fully dynamic scenes rendered at interactive frame rates.

The rest of this chapter is organized as follows. We first describe our implicit bounding plane representation before we show how to remove any remaining memory requirements in Sect. 7.1. Finally, we present a statistical evaluation of our approach for several test scenes (Sect. 7.4) and discuss our findings (Sect. 7.5).

7.1 Method

For better understanding, we decided to explain our methods in two parts. We first introduce the implicit bounding plane representation (Sect. 7.2) together with a description of the novel traversal (Subsec. 7.2.1) and construction (Subsec. 7.2.2). Afterwards, we go all the way and present our fully implicit representation (Sect. 7.3) and its updated traversal (Subsec. 7.3.1) and construction (Subsec. 7.3.2).

7.2 Implicit Bounding Plane Representation

Our main observation is that each bounding plane of a node in a BVH is defined by at least one scene primitive. In cases of polygons the plane is defined by a polygon vertex; for B-splines, by a control point; or by a bounding volume if instancing is used. For simplicity of explanation, we will concentrate on scenes solely composed of triangles.

Instead of saving each bounding plane of a node n_i explicitly, we save the scene primitives spanning n_i in the inner nodes. Using min/max operations on the bounds of the contained primitives, the AABB of each node can be recreated during traversal from only the six bounding triangles. As we keep primitives in inner and leaf nodes, the bounds of the child nodes do not necessarily share a common bounding plane, but the enclosing property of BVHs is still guaranteed, see Fig. 7.1 for a 2D example. All six primitives are contiguously mapped to memory and each such chunk is presorted so that the ordering in the triangle array corresponds to the nodes in the BVH. In cases where a single triangle spans more than one bounding plane of a node, e.g. triangle 4 and 7 in Fig. 7.1, less than six triangles are required to represent the bounds. To keep the memory layout consistent, we pad the node with the second closest triangle to the respective bounding plane of the same sub-tree.

Using a structure-of-arrays representation, we have two arrays, one containing the geometric information of the primitives, l_p, and one containing the child node indices l_c. Assuming the root node to have index 0, the index of the first bounding triangle p_i is derived from the child node index n_i by $p_i = n_i \cdot b$, where b is the number of bounding triangles per node. The original node index is used for storing the child pointer in l_c.

Hybrid BVHs conservatively estimate the AABB of a node by saving less than the standard six bounding planes and using an active ray interval. Following these approaches, we can choose an arbitrary number of bounding planes to represent a

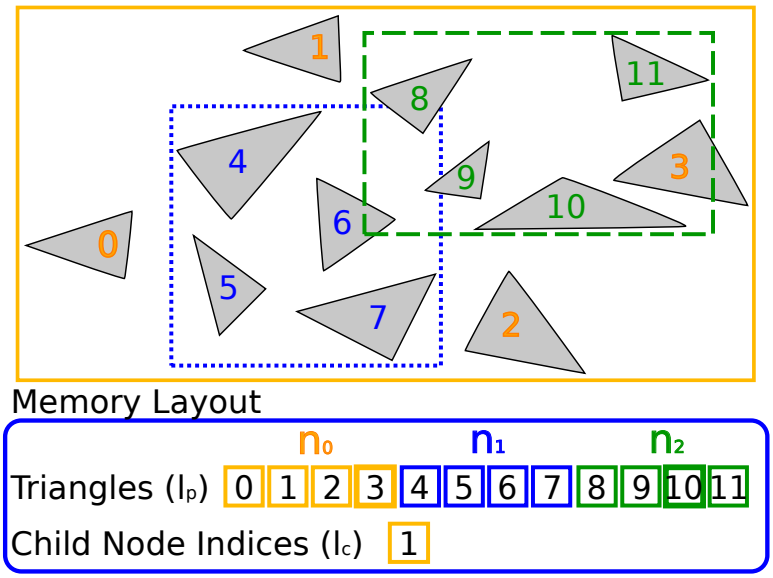


Figure 7.1: 2D example partitioning: The bounding boxes of the hierarchy are described solely by the vertices of the triangles. Only an array of triangles (l_p) and an array of offsets for the left child node (l_c) is saved.

node. For current standard (graphics) processors two opposing bounding planes per node seem to be the best choice in our setting, similar to [129, 134], see evaluation in Sect. 7.4. In the following, our description refers to the two triangle version. The single 4 byte child node index then encodes the following information: The lower two bits indicate the bounding axis that is spanned by the triangles (00: x, 01: y, 10: z) or whether it is a leaf node (case 11). We always use opposing bounding planes, therefore, the bounding axis is the one perpendicular to the bounding planes. As we map the left and right child next to each other in memory, the remaining 30 bits are used as the offset for the left child node only. For the leaf nodes, we use three bits to encode the number of triangles additionally contained in the node. During construction we ensure that only an even number of additional triangles is available in each leaf node as this allows us to encode up to fourteen additional primitives. Note that the count can be zero. The residual 27 bits encode the according offset into the triangle array which resides in memory right after all bounding triangles of the hierarchy. By sorting the children of each node according to their extent along the bounding axis we can incorporate ordered traversal [122] based on the ray direction.

7.2.1 Ray Intersection

Intersecting a ray with our implicit bounding plane representation is equivalent to a hybrid BVH traversal with an additional reconstruction and triangle intersection step. In each traversal step, we first compute the offset p_i of the first bounding triangle which is derived from the current traversal index n_i and we reconstruct the bounding planes. For this, we load only the data required for the current bounding axis, i.e. one float for each vertex of the two bounding triangles. After reconstructing and testing the minimum bounding plane, we test the maximum plane only if we found a valid intersection. If the intersection of a ray with the bounding planes is outside the active ray interval the sub-tree is skipped. Otherwise, the active ray interval is updated and the two bounding triangles are tested for intersection. Finally, we fetch the leaf node bits to test whether we reached a leaf node. Traversal either continues with the child nodes or in case of a leaf node the additional triangles are tested.

The chance of a hit with the bounding triangles in the first levels of the hierarchy is usually very low. It seems therefore beneficial to first test against the AABB of each triangle before testing it directly. Therefore, we could first reconstruct and test against the remaining bounding planes of each triangle along the current bounding axis. The vertex data of each further axis would only be loaded one axis at a time if the triangle was not already rejected beforehand. Only if a valid intersection for the complete AABB of the triangle is found the triangle itself would be tested. Even though this reduces the theoretical bandwidth requirements and the number of triangle intersections drops by approximately 50%, no real speed-up was experienced with the processor architectures we tested due to higher register pressure. However, this can be beneficial for future processor generations. Therefore, we test the triangles directly if the node is hit.

7.2.2 Construction

Most top-down BVH construction schemes can be directly applied to our representation. The only difference is an additional search step to find the bounding triangles spanning each node. These are excluded from further partitioning steps. Finding the bounding triangles requires a single scan over the active partition per node. The overall

complexity is then $O(n \log n)$, with n being the number of primitives. During construction and evaluation of the surface area heuristic (SAH) [70] it is important to keep in mind that a two-plane representation reduces the bounding volume only along a single dimension in each subdivision step. The bounding triangles are always chosen based on the partitioning axis of the parent node, as we can expect the largest surface reduction along this axis. We call the partitioning axis the one along which the triangles are subdivided into two new partitions and passed on to the child nodes.

7.3 Completely Implicit Representation

We now want to discard any explicit memory storage for the ADS at all. To remove the necessity for the bounding axis bits by, we use round-robin for choosing the axis, i.e. $xyzxyz\ldots$, depending on the depth in the tree. In order to be able to compute the children for any node, we enforce the hierarchy to be a complete, left-balanced tree arranged in breadth-first order. This allows us to index it like a heap without explicitly saving any pointers or indices [16]. For any implicit node n_i its children are indexed with $kn_i + m$ where k is the branching factor. In our case $k = 2$, and $m \in \{1, \ldots, k\}$ denotes the first child node, the second child and so on. By enforcing the hierarchy to be a complete tree, the leaf node property can be directly derived from the index, i.e. if the child index is larger than the number of implicit nodes in the scene a leaf has been reached. The last non-leaf node might have only one child instead of two, Fig. 7.2, as we only require the number of triangles in the scene to be even. In case of an odd number of triangles, the last one is replicated. We do not save any additional triangles in the leaf nodes, instead each primitive is a bounding primitive in some node of the hierarchy. Unfortunately, the compulsion of a complete tree forces us to use an object median split technique during construction.

7.3.1 Ray Intersection

Intersecting a ray with the completely implicit representation is similar to the approach in Subsec. 7.2.1 with few exceptions. Instead of testing for a leaf node, the current node traversal index is compared to the total number of nodes in the scene. Traversal is terminated if the index is larger. Otherwise, the child indices are computed and traversal continues.

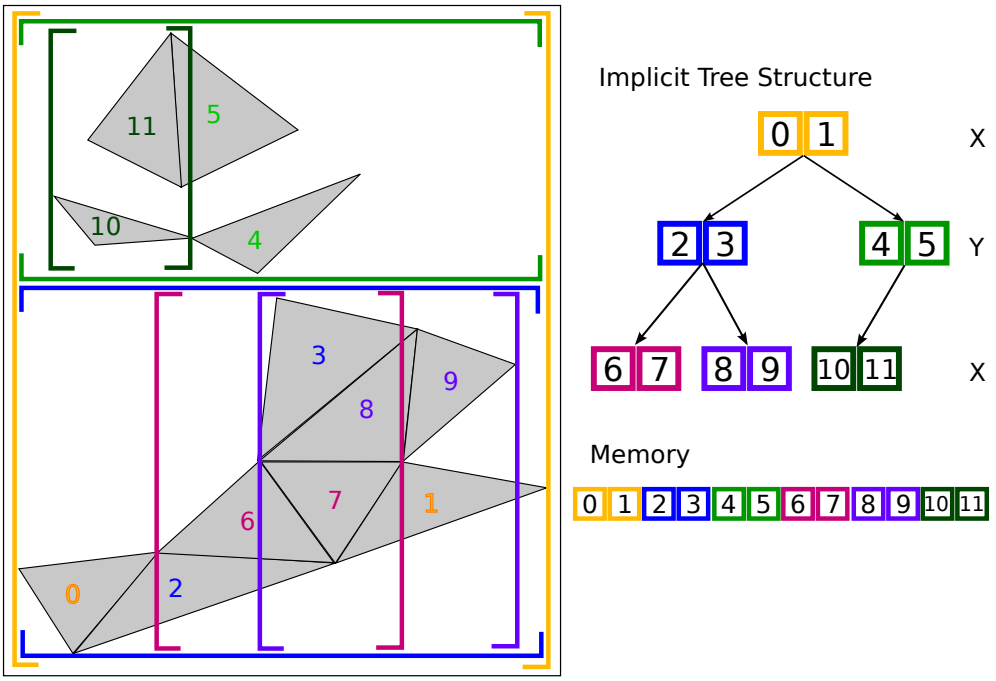


Figure 7.2: 2D example of the completely implicit object partitioning with three levels: The triangle arrangement implicitly describes a hierarchy. The bounds of each node are spanned by exactly two triangles. Left: Representation of the resulting bounding planes. The first and third level are bounded along the x-axis, the second level along the y-axis due to the round-robin scheme employed. The triangle index is colored according to the bounds the triangle represents. Top right: The scene triangles implicitly represent a complete binary tree of bounding planes. Bottom right: Representation in memory. Note that except for the triangles no additional memory is needed.

7.3.2 Construction

The properties of the completely implicit BVH open up the possibility for a parallel construction technique suitable for multiprocessor architectures. The hierarchy is built top-down and all nodes of one level are processed in parallel. In contrast to other approaches, parallelism in the upper nodes of the hierarchy is not enforced on a per node basis but threads operate across node boundaries, as will be described in the following.

As the bounding and partitioning axes are chosen in a round-robin fashion, see Sect. 7.3, all nodes of the same level in the hierarchy need to partition their enclosed primitives along the same axis. The partitioning axis is always equal to the bounding axis of the next hierarchy level. The hierarchical structure is known a priori due to the required left-balanced tree. We make use of an additional node index array I saving the currently active node a triangle might belong to and a split list S in which the starting index and the size of each active partition are saved.

The basic algorithm, as shown in Fig. 7.3, consists of four main steps for each level of the hierarchy. In the first step, all triangles and their according node indices are sorted along the current bounding axis. Then a stable sort using the node indices as keys is applied. While the first step sorts the triangles according to their spatial position, the second sort partitions the triangles according to their current node index without changing their respective order. This puts the minimum bounding triangles at the correct positions and allows for a direct partitioning of the active nodes for further subdivisions. We then search for the maximum bounding triangle in each split and swap it to the second position in the split. We update the node indices for the next iteration (details are given below), remove the old splits, as they are already processed, and emit new splits for each node of the next level. The process is repeated until no split contains more than two triangles. We do not sort the triangles and indices directly but rather utilize a permutation array for efficiency. The memory requirement for the parallel construction is $O(n)$ as we need one integer per triangle plus the split list, which is of the size $n/4$ at most.

The following describes the construction process in more detail for reimplementation. The construction algorithm for each level of the hierarchy consists of four main components: a lexicographical sort, a maximum triangle search, an update of the split

```
axis = 0;
S = { (0,n) }; // Split list
I = {0,0,0,...,0}; // Index list
parallel_construction(triangles,S,I,axis);

void parallel_construction(triangles,splits, I, axis){
for all levels of the hierarchy{
lexicographicalSort(triangles, S[0][0], I, axis);
if(lastLevelreached){ return; }
maximumTriangleSearch(triangles,S,axis);
updateIndices(I, S);
createNewSplitsFromOld(S);
axis = (axis + 1) % 3;
}}
```

Figure 7.3: Pseudo-code of the parallel construction scheme for the completely implicit representation.

indices, and spawning of new splits from the previous ones, Fig. 7.3. We start with a single split at index 0 with a size n equal to the number of scene primitives and initialize I to zeros. The algorithm then loops over all $\lfloor \log_2(n/2) \rfloor + 1$ levels of the hierarchy. In each loop we first apply the lexicographical sort to all triangles, i.e. sort them according to their spatial position and then a stable sort on the node indices is applied. Already finalized triangles in front of the first split are excluded. The sort swaps already finished nodes to the front and sorts all triangles of the same node along the bounding axis. The remaining triangles in each node are split into its two child nodes where each child already has the minimum triangle at the correct position. Next, we search for the bounding triangle of the maximum bound in the remainder of the child triangles and swap it with the second position in each split. We can use a simple swap operation instead of shifting the triangles over to the end of the split since we only required the triangles to be sorted for the actual split operation. As long as the average over all splits of a given level holds more than 4 triangles, we assign *averageTrianglesPerSplit* / 4 threads to each split. Since for the maximum search each thread will find its own maximum, we use atomic compare and swap (atomicCAS) functions in case a new maximum was found to ensure the overall maximum is found. The threads are assigned in reverse order per split to minimize the warp serialization

due to the atomic operations. As soon as the average number of triangles per split falls down to 4 triangles, we only use a single thread per split and can therefore switch to a kernel without atomic operations.

For each split the algorithm now updates the node index values. The first two indices of each open split S_i are assigned a value of $idx = finalized + i$ where $finalized = 2^{lvl} - 1$ is the number of already correctly created nodes. i is the index of the split in the split list and lvl the current level of the hierarchy. The value of the other triangles in a split are set to $2\ idx + 1$ for their respective splits which is the index of the left child. We use the same thread distribution for each split as described in the last paragraph for parallelism.

We remove the active splits and insert new splits for the left and right child nodes into the queue if they contain two or more triangles. Let $numSplits$ be the number of the old open splits, pos_i the starting position of the i^{th} split and num_i its size. The size num_L and num_R for the new splits is chosen in a way to guarantee a left-balanced, complete tree.

$$\begin{aligned} half_i &= \frac{num_i}{2} \\ H &= \lfloor \log 2\,(half_i) \rfloor \\ num_R &= 2\left((2^{H-1}-1) + \max(0, half_i - 3 \cdot 2^{H-1} - 3)\right) \\ num_L &= num_i - 2 - num_R \end{aligned} \tag{7.1}$$

where H is the depth of the tree.

The positions of the new left and right splits resulting from pos_i are computed by

$$\begin{aligned} pos_L &= pos_i + 2(numSplits - i) \\ pos_R &= pos_L + num_L \end{aligned} \tag{7.2}$$

The computed positions of the splits are already at the positions that are needed *after* the next lexicographical sort. Finally, the bounding axis is incremented to the next level.

This procedure automatically builds a breadth-first tree. An example for the first three levels of a scene with twelve triangles is given in Fig. 7.4.

	Splits		S0										
	Triangles		7	5	0	8	2	1	3	4	11	6	9	10
	Node index		0	0	0	0	0	0	0	0	0	0	0	0
Lvl0	Splits	sort + max search	S0											
	Triangles		0	1	2	11	6	5	4	3	8	7	9	10
	Node index		0	0	0	0	0	0	0	0	0	0	0	0
	Splits	update node indices	S0											
	Triangles		0	1	2	11	6	5	4	3	8	7	9	10
	Node index		0	0	1	1	1	1	1	1	1	1	1	1
	Splits	new splits			S0						S1			
	Triangles		0	1	2	11	6	5	4	3	8	7	9	10
	Node index		0	0	1	1	1	1	1	1	1	1	1	1
Lvl1	Splits	sort + max search			S0						S1			
	Triangles		0	1	2	3	7	6	8	9	4	5	10	11
	Node index		0	0	1	1	1	1	1	1	1	1	1	1
	Splits	update node indices			S0						S1			
	Triangles		0	1	2	3	7	6	8	9	4	5	10	11
	Node index		0	0	1	1	3	3	3	3	2	2	5	5
	Splits	new splits							S0		S1		S2	
	Triangles		0	1	2	3	7	6	8	9	4	5	10	11
	Node index		0	0	1	1	3	3	3	3	2	2	5	5
Lvl2	Splits	sort + max search							S0		S1		S2	
	Triangles		0	1	2	3	4	5	6	7	8	9	10	11
	Node index		0	0	1	1	2	2	3	3	3	3	5	5
...														

Figure 7.4: Example of the first three levels in the parallel hierarchy creation process for the completely implicit representation. The spatial arrangement of the triangles according to the construction is shown in Fig. 7.2.

7.4 Results

We evaluate our presented algorithm on several scenes with varying complexity, including ones with high triangle count (Thai Statue), teapot-in-a-stadium problems (Fairy), largely differing scene primitives (Crytek Sponza), animation (Fairy, Breaking Lion) and problematic scenes for object median cuts (Venice and Crytek Sponza), or combinations of these scene attributes. To evaluate the influence of the implicit bounding plane representation, we show results for both the Implicit Object Space Partitioning with 4 bytes (IOSP-4) and the complete implicit representation (IOSP-0). We also implemented a hybrid version that saves the top-levels as uncompressed BVH nodes using a SAH builder where each leaf points towards a separate IOSP-0 (2-Lvl IOSP). We analyse and discuss our optimizations, bandwidth considerations, incoherent rays as encountered in global illumination simulations, construction performance, as well as the two-level approach for increased performance.

We have produced both a CPU variant and a GPU implementation using NVIDIA CUDA. All statistics were measured on a system with an Intel Core i7-2600 with 3.4 GHz, 16GB RAM, and an NVIDIA GeForce GTX 580 with 3GB of memory, running on a 64-bit Windows system. All results are produced at a resolution of 1024×768 pixels if not stated otherwise.

For a general comparison, if appropriate, we make use of a BVH implementation using the surface area heuristic - BVH(SAH) - and using an object median split - BVH(OMS). In accordance with [120] we use a binning approach with ten bins during construction for evaluation of the SAH. We impose a minimum triangle count of four triangles per leaf node. The same strategy was used for our IOSP-4 and 2-Lvl IOSP. The associated statistics are given in Table 7.5.

Number of Bounding Triangles We first verified our choice of using only two boundary triangles by comparing performance for different numbers of bounding triangles for the IOSP-0. For one bounding triangle, we follow the approach of [28] where the single bound encodes the half-space in which the geometry resides. We extend the round-robin scheme so that for the first three levels the maximum bounds are saved for the left child nodes (respectively the minimum bounds for the right children) and the minimum bounds for the next three levels (respectively the maximum bounds for

the right children). For the six triangles case, a complete AABB is reconstructed in each traversal step. For current standard (graphics) processors, choosing two bounding triangles per node resulted in the best performance in our test scenes, Table 7.1. This may change in future hardware with larger cache lines or higher costs per memory access compared to the computational power. Using less bounding triangles per node would require empty nodes for efficiency [117], while using more causes a too high computational load on current processors. In the following experiments, we always used the version with two bounding triangles.

Scene	1 (CPU)	2 (CPU)	6 (CPU)	1 (GPU)	2 (GPU)	6 (GPU)
Breaking Lion	0.629s	0.489s	0.639s	0.094s	0.077s	0.060s
Crytek Sponza	1.972s	1.614s	7.383s	0.144s	0.085s	0.268s
Fairy	1.710s	1.077s	1.996s	0.131s	0.060s	0.104s
Robot Girl	0.969s	0.718s	0.922s	0.083s	0.045s	0.050s
Thai Statue	0.819s	0.383s	0.933s	0.153s	0.050s	0.103s
Venice	3.068s	2.102s	3.694s	0.248s	0.164s	0.176s

Table 7.1: Evaluation of the influence of bounding triangles per node for primary rays. CPU and GPU traversal time are given in seconds.

Bandwidth Considerations We measured the bandwidth requirements assuming a perfect memory access and tracing one ray after the other, i.e. no caching. Each tested bounding box of a BVH is assumed to be 32 bytes in size, while each tested triangle is counted as 36 bytes (nine float values for the three vertices). Additional data like texture coordinates, normals etc. are not included, as these are accessed only in the shading step, which is the same for all tested approaches. Statistics are given in Table 7.5.

Compared to the BVH(SAH) the theoretical bandwidth increases by a factor of 2.77 to 8.35 with 5.03 on average for the IOSP-0, 1.49 to 2.04 with 1.78 on average for the IOSP-4, and a factor of 1.09 to 3.05 with 1.69 on average for a two-level IOSP-0 with 15 uncompressed top-levels. In practice the values will vary depending on the hardware capabilities, like cache line size and traversal technique used.

Incoherent Rays One of the main advantages of ray tracing is that it can employ secondary rays to compute effects such as global illumination, soft shadows, reflection or

Scene	(#B)	BVH(SAH)	IOSP-4	IOSP-0
Breaking Lion	1	69.391	39.322	19.181
Crytek Sponza	1	32.319	12.788	2.844
Fairy	1	61.280	29.127	5.761
Robot Girl	1	85.020	49.152	19.784
Thai Statue	1	73.156	21.845	11.523
Venice	1	41.391	18.614	3.456
Breaking Lion	3	93.437	56.510	26.963
Crytek Sponza	3	25.233	9.180	1.405
Fairy	3	62.915	29.677	4.575
Robot Girl	3	106.63	63.550	28.468
Thai Statue	3	88.612	29.263	15.271
Venice	3	41.665	18.559	3.299

Table 7.2: Influence of the number of bounces (#B) in a path tracing simulation according to the number of bounces on the GPU. Numbers are given in million rays per second. Computations include ray generation, traversal, shading, and texturing. The first three scenes contain 2 light sources each, while the latter three have 1 light source.

refraction. The incoherency of these rays, especially in Monte-Carlo simulations, poses problems on the efficiency of ray tracers due to incoherent memory access and diverging traversal paths, especially on a highly parallel processor as the GPU. Table 7.2 shows the results of our test scenes rendered with up to three light bounces. One ray path per pixel is created using pure random sampling over the pixel domain and the hemisphere domain (to increase incoherency) and one shadow ray is traced for each light source at each path vertex.

Animations For animated and dynamic scenes not only traversal performance but also construction times are of importance. Here, we analyse our construction technique for the IOSP-0 from Subsec. 7.3.2. All experiments were conducted directly on the GPU. In a straightforward implementation, we would simply assign a single thread to each split at all levels. However, this does not create enough parallelism at the top levels of the hierarchy. An alternative is to assign one thread to each triangle and search for the split that this triangle falls into. Obviously, this will cause issues at the lower levels of the hierarchy as the number of splits to search for doubles for each

Scene	1 Thread/Tri	1 Thread/Split	Adaptive
Fairy	0.603s	0.284s	0.159s
Breaking Lion	>10s	1.533s	0.226s

Table 7.3: Comparison of the construction times using one thread per triangle for all levels of the hierarchy (1 Thread/Tri), one thread per split (1 Thread/Split) for all levels, and our adaptive approach that uses multiple threads per split.

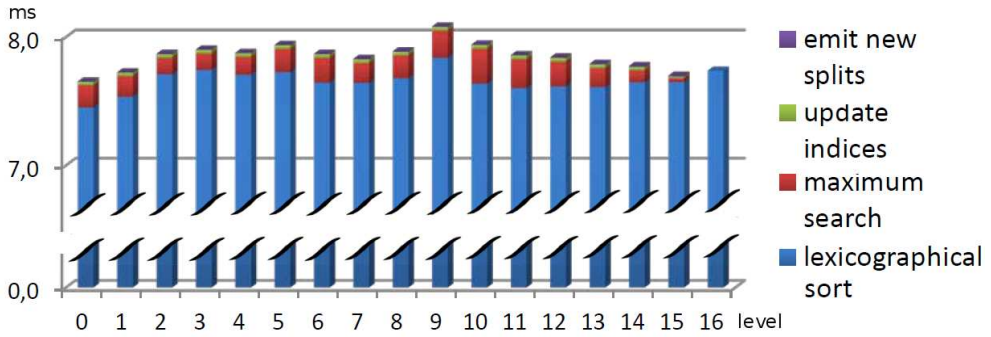
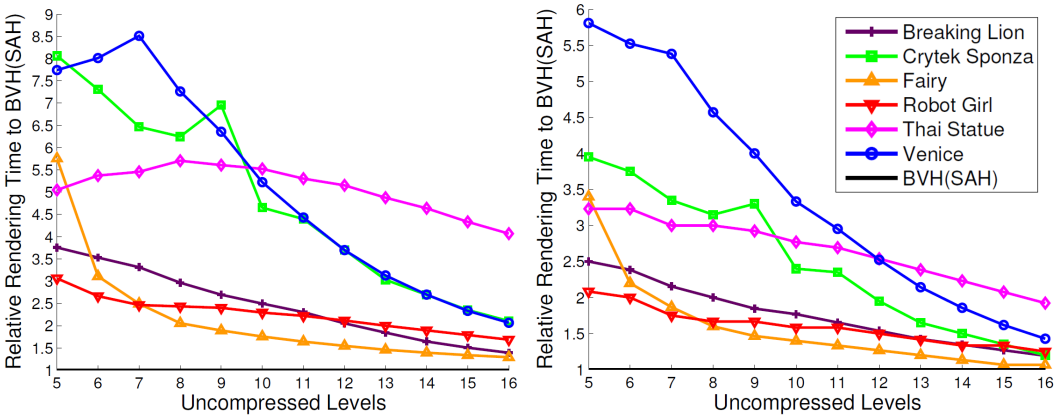


Figure 7.5: In order to evaluate our parallel construction algorithm, we show the time needed by the different steps of our construction per level for the animated scene FAIRY consisting of 174k triangles. The timings include the lexicographical sort (blue), updating the node indices (green), creating the new splits (violet), and the search for the maximum triangle (red).

level. In Table 7.3, we show a comparison of construction timings between creating the hierarchy using one thread per triangle, one thread per split and our adaptive approach. In the adaptive approach we assign multiple threads to each split, so that the average number of triangles per thread is limited to the same value at all levels of the hierarchy. Naturally, this will revert to one thread per split at the bottommost levels of the hierarchy. Our adaptive approach reduces construction time for the animated scenes up to 85% compared against the straightforward implementation, Table 7.3.

Fig. 7.5 illustrates the construction performance in detail for each level. Our algorithm shows a virtually constant construction time per level. About 95% of the time is used by the lexicographical sort for which we used the CUDA Thrust library.

Two-level approach Representing the important top-levels of the hierarchy in an uncompressed BVH format and using the compressed representation for the lower levels is an established technique to provide a convenient trade-off between performance and



(a) CPU **(b)** GPU

Figure 7.6: Comparison of render times using different numbers of levels for the uncompressed BVH.

memory requirements for several compression schemes [5, 37, 63, 64, 83, 104]. We analysed the influence of the ratio between uncompressed levels and compressed levels in terms of ray tracing performance in Fig. 7.6. For 15 uncompressed levels the memory requirements are only up to 1MB for our 2-Lvl IOSP while performance is between 23-74% for the CPU and 48-93% for the GPU compared to the BVH(SAH), depending on scene complexity, Table 7.5. The two-level approach works best for non-uniform triangle distributions. For the THAI STATUE a median cut in the upper levels is of similar quality compared to a cut based on the SAH, therefore only a relatively small speed-up is achieved.

Comparison to other memory reduction techniques In the following we compare our technique to more sophisticated techniques than a standard BVH. Reusing shared bounding planes [32, 33, 51] reduces memory and bandwidth requirements in a BVH by 43–50% and 35–38%, respectively, without a negative influence on the rendering times [33]. The bounding interval hierarchy [116] performs up to par to an optimized BVH or kd-tree but the memory requirements are only ten bytes on average with a careful implementation (69% reduction). If memory reductions of 50% to 70% are sufficient, these techniques allow for performance similar to a BVH.

In Table 7.4 we compare our technique with the divide-and-conquer (DAC) approach by Mora et al. [79] which also does not require to save any acceleration data structure.

Scene	Ours (CPU) Single	Ours (CPU) Packets	Ours (GPU) Single	[79] (CPU) Packets
Fairy	0.60	14.28	10.4	6.8
Thai Statue	2.35	3.37	12.05	1.28

Table 7.4: Comparison to Mora et al. [79]. Resolution is 1024×1024 pixels, only primary rays are traced, and simple eye shading used. Frames per second are reported. Single = Single ray tracing. Packets = Packet tracing.

The resolution was set to 1024×1024 pixels and only primary rays were traced. We chose the Fairy and Thai Statue scene as they provide the best insights into the strength and weaknesses of both approaches. Please note that the comparison has to be done carefully as the processor architectures differ. In [79] an Intel-core 2 duo E6850 with 3 GHz was used while we use a Core i7-2600 with 3.4 GHz. Also note that the DAC uses conic packets for primary rays resulting in an additional speed-up of factor 1.3 – 3.8 depending on the scene (speed-up taken from Figure 8 in [79]). As expected DAC achieves better performance on the CPU for smaller scenes, probably due to the spatial median split employed which provides drastically better clipping quality than the object median split required by our technique. For larger scenes the overhead due to the triangle streaming in the DAC approach becomes more apparent. Our technique can be easily ported to the GPU where we achieve speed-ups between a factor of 1.5 and 9.4 compared to DAC. No packet tracing was used in our GPU timings which would further increase performance, especially since packets are more robust to non-optimal subdivision schemes in terms of performance.

7.5 Discussion

Overall, the IOSP shows impressive performance results considering it is a complete implicit acceleration structure and, even in its simplistic form, enables interactive ray tracing. Currently, the most-limiting factor for faster traversal performance on GPUs seems to be high memory bandwidth due to inferior clipping for the object median cut. The object median cut partitioning scheme is known to be inferior to other tree structures [115], but is currently a necessity for the implicit child index computation.

Method	N_T	N_I	R	BW/frame	Mem
Scene - Breaking Lion - 1,604,054 triangles, 96.331 MB of memory used for geometry					
BVH(SAH)	26,577k	5,678k	35.747 Mrays/s	0.983 GB	33.524 MB
BVH(OMS)	58,942k	10,384k	18.289 Mrays/s	2.105 GB	33.554 MB
IOSP-4	29,503k	32,975k	20.696 Mrays/s	1.469 GB	2.694 MB
IOSP-0	59,435k	62,471k	10.213 Mrays/s	2.725 GB	0 MB
2-Lvl IOSP (15)	28,624k	13,087k	23.831 Mrays/s	1.067 GB	1.001 MB
Scene - Crytek Sponza - 279,163 triangles, 19.587 MB of memory used for geometry					
BVH(SAH)	80,626k	6,814k	41.391 Mrays/s	2.631 GB	5.608 MB
BVH(OMS)	289,334k	37,531k	11.398 Mrays/s	9.881 GB	5.283 MB
IOSP-4	77,406k	85,146k	26.214 Mrays/s	3.819 GB	0.471 MB
IOSP-0	260,178k	278,989k	9.252 Mrays/s	12.051 GB	0 MB
2-Lvl IOSP (15)	128,160k	49,780k	27.778 Mrays/s	4.643 GB	0.535 MB
Scene - Fairy - 174,117 triangles, 12.365 MB of memory used for geometry					
BVH(SAH)	47,680k	5,349k	56.174 Mrays/s	1.600 GB	3.577 MB
BVH(OMS)	191,754k	22,226k	20.165 Mrays/s	6.460 GB	4.194 MB
IOSP-4	58,461k	66,259k	32.768 Mrays/s	2.936 GB	0.294 MB
IOSP-0	186,332k	194,406k	13.107 Mrays/s	8.510 GB	0 MB
2-Lvl IOSP (15)	52,533k	9,973k	46.875 Mrays/s	1.738 GB	0.582 MB
Scene - Robot Girl - 1,010,054 triangles, 60.653 MB of memory used for geometry					
BVH(SAH)	25,693k	2,832k	71.494 Mrays/s	0.861 GB	21.625 MB
BVH(OMS)	143,301k	27,639k	18.289 Mrays/s	5.197 GB	16.777 MB
IOSP-4	39,033k	42,496k	39.322 Mrays/s	1.910 GB	1.710 MB
IOSP-0	114,559k	119,955k	17.476 Mrays/s	5.242 GB	0 MB
2-Lvl IOSP (15)	32,457k	14,857k	46.875 Mrays/s	1.206 GB	0.513 MB
Scene - Thai Statue - 10.000.002 triangles, 640.002 MB of memory used for geometry					
BVH(SAH)	21,031k	3,708k	65.536 Mrays/s	0.751 GB	212.332 MB
BVH(OMS)	57,481k	5,607k	37.449 Mrays/s	1.901 GB	237.348 MB
IOSP-4	29,663k	35,532k	26.214 Mrays/s	1.536 GB	16.792 MB
IOSP-0	51,519k	52,454k	15.729 Mrays/s	2.324 GB	0 MB
2-Lvl IOSP (15)	57,723k	36,169k	27.778 Mrays/s	2.293 GB	0.520 MB
Scene - Venice - 2.447.208 triangles, 192.161 MB of memory used for geometry					
BVH(SAH)	46,617k	5,477k	39.332 Mrays/s	1.573 GB	49.473 MB
BVH(OMS)	314,809k	39,094k	8.278 Mrays/s	10.693 GB	55.957 MB
IOSP-4	66,098k	71,579k	21.845 Mrays/s	3.229 GB	4.130 MB
IOSP-0	285,213k	302,466k	4.795 Mrays/s	13.136 GB	0 MB
2-Lvl IOSP (15)	73,191k	34,957k	22.059 Mrays/s	2.754 GB	0.938 MB

Table 7.5: Comparison of IOSP-4, IOSP-0 and 2-lvl IOSP (with 15 uncompressed top-levels) with a SAH-BVH (BVH(SAH)) and an object median Split BVH (BVH(OMS)). N_T/N_I is the number of tested nodes/ray-object intersections in total, R is the number of traversed rays (only primary) in millions per second on the GPU, excluding construction. BW/frame is the minimal necessary data throughput (bandwidth) per frame, and Mem is the memory usage of only the AS in megabytes.

Another angle to decrease the overall bandwidth requirements could possibly be to apply fast mesh compression techniques [94].

Our approach suffers from the same drawbacks as most object space partitioning schemes when encountering a mixture of small and large primitives in a scene. Larger primitives are kept higher up in the hierarchy which is generally beneficial for some scenes [47, 134], but incorporating early split clipping [30] is an important challenge for future work and improved performance. This was the main reason why we decided to explicitly save the child index in our IOSP-4 approach, which allows us to incorporate more sophisticated building heuristics. Another obvious limitation is that only primitive types that are not based on connectivity can be used. This means that re-ordering of the primitives needs to be possible without destroying the memory layout of the underlying geometry.

Our focus is on computing ray intersections with the scene, but collision detection is another possible application for the IOSP. However, AABBs are usually not the bounding volume of choice for this task, and a direct application of our approach is difficult due to the triangles in the inner nodes of the hierarchy.

This concludes Part III of the thesis. In this part, we presented a memoryless acceleration approach for ray tracing which leads two significant memory reduction by up to two orders of magnitude. The implicit approach is well-suited for many-core architectures and enables interactive ray tracing on commodity hardware. A 2-level variant allows for a convenient trade-off between performance and memory reduction. While several limitations still exist, we see promising prospects for further research and improvements due to our novel way of approaching the issue.

Part IV

Conclusion

8

Summary and Future Work

We will conclude the thesis by summarizing our contributions and by discussing promising directions for future work. We addressed two major issues of MC rendering: noise due to insufficient sampling, and memory consumption of acceleration structures. Both are major issues in today's ray tracing pipelines and our contributions are unified under the general idea of accelerating MC rendering.

In Part II, we first proposed two different filtering approaches for advanced denoising. We first presented a novel approach for efficiently removing MC noise in images rendered with global illumination techniques in conjunction with depth-of-field at very low sampling rates. The proposed approach is a complete post-process and is applicable to standard MC renderers without invasive changes to the rendering algorithm itself. Another benefit is that it is completely parallelizable and maps directly to today's GPU cards. The findings from Chap. 4 indicate that the method generates visually plausible results and is suitable for interactive or even real-time ray tracing applications. In Chap. 5, we have presented a way to estimate the error of arbitrary reconstruction techniques for MC renderings and how to exploit this estimate to select a close-to-optimal filter per pixel for adaptive reconstruction. We introduced the idea of filter caches, sparse high-quality radiance estimates within the image plane, to robustly estimate the error. Our adaptive cache placement is based on local variances of the filter bank as well as the MC estimator to conservatively reducing the threat of choosing wrong filters at crucial image positions. The freedom to basically combine any reconstruction technique with our robust error estimator significantly improves the reconstruction results

over state-of-the-art approaches. Still, it introduces only little overhead to the overall rendering pipeline.

In Part III, we introduced a novel concept for removing the memory requirements of acceleration structures for ray tracing. We presented a completely implicit object space partitioning scheme, Chap. 7, which requires no additional memory at all. Our approach is based on the insight that the bounding planes of a hierarchical acceleration data structure can be represented and accessed efficiently by geometry presorting. This novel design has several benefits which distinguishes it from the other methods that remove the acceleration structure completely. While it is perfectly suitable for many-core architectures (e.g. GPUs), it does not require to process ray batches and must be created only once per time step of an animation, independent of the viewpoint or lighting condition since it is statically represented by the underlying geometry. By using a 2-level approach, we found that the evident performance loss that originates from the memoryless representation can be partially compensated and a trade-off between performance and memory reduction is conveniently possible. We showed that the presented approach still allows for interactive ray tracing on commodity hardware, even for dynamic scenes.

Finally, we wish to discuss promising aspects for future work. A known limitation of image-space filtering for noise removal of MC renderings is that these methods tend to over-smooth regions with specular or highly glossy materials. We experienced this behavior also with our Sample-Based Manifold Filtering from Chap. 4. A possible solution to this is to separate the rendering output into different meaningful layers, filter each according to its specific characteristics, and to composite the layers again afterwards. A faster alternative idea would be to filter the complete image but use these different layers as guidance or features to distinguish pixel similarity. Another issue that we have not yet considered in the Sample-Based Manifold Filtering is the incorporation of motion blur effects. While the original sweep-blur algorithm [107] shows that the approach already can handle motion blur, it is not clear how to associate motion samples in the manifold-based approach. However, we expect that the concept of improving samples locally before distributing them on the image plane could also be beneficial for motion blur situations.

For all our filtering approaches, we currently do not consider temporal coherency over multiple frames, e.g. in animation, besides using a fixed seed for generating the samples on the image plane and the lens. Hence, temporal coherence is not guaranteed. While we did not observe problems in in-focus regions, it can cause flickering at edges and in the transition areas between in-focus and out-of-focus regions. The Sample-based Manifold Filtering can potentially be improved by manifold reuse over multiple frames which, however, is a topic that needs further investigation. An interesting step in this direction for our error analysis framework, Chap. 5, might be to extend our graph-cut technique to the time dimension.

So far, our filtering techniques has mostly been used in conjunction with path tracing. We believe that building on stochastic ray tracing techniques in general is a future oriented rendering paradigm, and other approaches such as bidirectional path tracing [62] or Metropolis light transport [114] can also greatly benefit from our contributions.

The concept of *Implicit Object Space Partitioning*, Chap. 7, is a promising and interesting approach to memoryless acceleration. Implicit acceleration data structures have only recently gained attention in the rendering community, and we hope that we can inspire follow-up work in that field with our novel concept. Finding a solution for an implicit representation with an arbitrary splitting scheme is an open problem. For comparable performance to state-of-the-art techniques, integration of spatial splits

[110] would be a necessity, but the requirement for multiple object references seems problematic for an fully implicit representation. Another fruitful direction might be to investigate if spatial partitioning schemes can be implicitly represented without a lazy evaluation scheme (e.g. divide-and-conquer ray tracing). We investigated only triangles as the basic primitive, but other scene representations are possible. A dedicated hardware implementation of our IOSP-0 is also a promising direction, as the main ingredients are a sorting procedure and triangle intersections. Both can be efficiently implemented in hardware [58, 129]. These are exciting avenues for future work.

Part V

Appendix

9

Abbreviations

General	
GI	Global Illumination
DoF	Depth of Field
MB	Motion Blur
MC	Monte Carlo
BSDF	Bidirectional Scattering Distribution Function
BRDF	Bidirectional Reflectance Distribution Function
BTDF	Bidirectional Transmittance Distribution Function
MSE	Mean Squared Error
SSIM	Structural Similarity Index Measurement
spp	Samples Per Pixel
AMF	Adaptive Manifold Filtering
SBMF	Sample-Based Manifold Filtering
GREE	General and Robust Error Estimation
AS	Acceleration Structure
BVH	Bounding Volume Hierarchy
AABB	Axis-aligned Bounding Box
OSP	Object Space Partitioning
SAH	Surface Area Heuristic
IOSP	Implicit Object Space Partitioning

Table 9.1: Abbreviations used in the thesis.

10

Notation

This chapter contains notation tables for general notations used throughout the thesis and our work on denoising, Part II. We omit notation tables for Part III as it contains only a few mathematical equations and definitions.

General	
N	Noisy image.
F	Filtered output image.
i	Index of the $i-th$ element (usually pixel index).
$L_i(x, \omega_i)$	Incident radiance.
$L_o(x, \omega_o)$	Exitant radiance.

Table 10.1: General notations used throughout the thesis.

Sample-based Adaptive Manifolds (Sect. 4.2)	
K	Number of manifolds.
η^k	$k-th$ manifold.
$s^c_m(i)$	Colors of the m-th sample at pixel i.
$s^g_m(i)$	Guide values of the m-th sample at pixel i.
$s^c(i)$	d_N-sized vector with sample colors of pixel i.
$s^g(i)$	d_N-sized vector with sample guides of pixel i.
$M^k_{\text{splat}}(i)$	Splatted value on the i-th pixel of the k-th manifold.
$M^k_{\text{splatweight}}(i)$	Splatting weight.
$M^k_{\text{blur}}(i)$	Blurred value on the manifold.
$M^k_{\text{blurweight}}(i)$	Blurred, splatting weight.
ϕ	Gaussian kernel to control the splatting falloff.
$G_m(i)$	Filtered output of the m-th sample in the i-th pixel.
$w^k_m(i)$	Weight of the m-th sample in the i-th pixel on the k-th manifold.
Fast Sweep-Blur (Sect. 4.3)	
n_F	Number of filter scales.
n_L	Number of linear manifolds.
$B^k_f(i)$	k-th manifold blurred with the filter scale f.
$A^k(i)$	Contribution of the manifold k for the pixel i.
f^k	Filter scale of the manifold k.
cov^k_f	Coverage value for manifold k for the filter scale f.

Table 10.2: Notation used in Chap. 4.

General and Robust Error Estimation (Sect. 5.1)	
$\mathbb{F}$	Filter bank.
$\mathbf{L}$	Label map.
$\mathbf{F}_{\sigma^2}$	Variance of the filter bank.
$\mathbf{N}_{\sigma^2}$	Variance of the noisy image.
$\mathbf{D}$	Errormap (dense).
s	Sparsity.
b	Sample budget.
c	Cache sampling rate.
n	Noisy sampling rate.
m	Number of caches.

Table 10.3: Notation used in in Chap. 5.

11

Credits

I would like to credit all people and institutes which provided images, scenes and 3D data sets for this thesis, in particular the Stanford 3D Scanning Repository, the Cornell University Program of Computer Graphics, the Utah 3D Animation Repository, Guillermo M. Leal Llaguno of Evolución Visual for the San Miguel scene, Marko Dabrovic and Mihovil Odak for the Sibenik cathedral, and Anat Grynberg and Greg Ward for the conference room. Finally, I would like to thank the computer graphics community and all researchers who inspired this thesis. Thank you!

Bibliography

[1] Agarwala, A., Dontcheva, M., Agrawala, M., Drucker, S., Colburn, A., Curless, B., Salesin, D., and Cohen, M. 2004. Interactive digital photomontage. In *ACM SIGGRAPH 2004 Papers*. SIGGRAPH '04. 294–302.

[2] Aila, T. and Laine, S. 2009. Understanding the Efficiency of Ray Traversal on GPUs. In *Proc. of High-Performance Graphics*. 145–149.

[3] Bauszat, P., Eisemann, M., Eisemann, E., and Magnor, M. 2015. General and robust error estimation and reconstruction for monte carlo rendering.

[4] Bauszat, P., Eisemann, M., John, S., and Magnor, M. 2014. Sample-based manifold filtering for interactive global illumination and depth of field. *Computer Graphics Forum 34,* 1 (Nov.), 265–276.

[5] Bauszat, P., Eisemann, M., and Magnor, M. 2010. The Minimal Bounding Volume Hierarchy. In *Proc. of Vision, Modeling, and Visualization (VMV'10)*. Siegen, Germany, 227–234.

[6] Bauszat, P., Eisemann, M., and Magnor, M. 2011. Guided image filtering for interactive high-quality global illumination. *Computer Graphics Forum (Proc. of Eurographics Symposium on Rendering (EGSR)) 30,* 4 (June), 1361–1368.

[7] Bennett, E. P., Mason, J. L., and McMillan, L. 2007. Multispectral bilateral video fusion. *IEEE Transactions on Image Processing 16,* 5, 1185–1194.

[8] Bertalmio, M., Sapiro, G., Caselles, V., and Ballester, C. 2000. Image inpainting. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '00. 417–424.

[9] Bhat, P., Zitnick, C. L., Snavely, N., Agarwala, A., Agrawala, M., Cohen, M., Curless, B., and Kang, S. B. 2007. Using photographs to enhance videos of a static scene. In *Proceedings of the 18th Eurographics Conference on Rendering Techniques*. EGSR'07. 327–338.

[10] Boufounos, P., Duarte, M. F., and Baraniuk, R. G. 2007. Sparse signal reconstruction from noisy compressive measurements using cross validation. *Statistical Signal Processing, IEEE/SP Workshop on 0*, 299–303.

[11] Boykov, Y., Veksler, O., and Zabih, R. 2001. Fast approximate energy minimization via graph cuts. *IEEE Trans. Pattern Anal. Mach. Intell. 23,* 11, 1222–1239.

[12] Buades, A., Coll, B., and Morel, J.-M. 2005. A non-local algorithm for image denoising. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. CVPR '05. 60–65.

[13] Chan, T. F. and Shen, J. 2002. Mathematical models for local nontexture inpaintings. *SIAM J. Appl. Math 62*, 1019–1043.

[14] Chen, J., Paris, S., and Durand, F. 2007. Real-time edge-aware image processing with the bilateral grid. *ACM Trans. Graph. 26,* 3, 1–10.

[15] Chen, J., Wang, B., Wang, Y., Overbeck, R. S., Yong, J.-H., and Wang, W. 2011. Efficient depth-of-field rendering with adaptive sampling and multiscale reconstruction. *Comput. Graph. Forum 30,* 6, 1667–1680.

[16] Cline, D., Steele, K., and Egbert, P. 2006. Lightweight Bounding Volumes for Ray Tracing. *Journal of Graphic Tools 11,* 4, 61–71.

[17] Cook, R. L., Porter, T., and Carpenter, L. 1984. Distributed ray tracing. In *Proc. of the 11th annual conference on Computer graphics and interactive techniques*. SIGGRAPH '84. ACM, New York, NY, USA, 137–145.

[18] Crow, F. C. 1984. Summed-area tables for texture mapping. In *Proc. of the 11th annual conference on Computer graphics and interactive techniques*. SIGGRAPH '84. 207–212.

[19] Dabov, K., Foi, A., Katkovnik, V., and Egiazarian, K. 2006. Image denoising with block-matching and 3d filtering. In *IN ELECTRONIC IMAGING, PROC. SPIE 6064, NO. 6064A-30*.

[20] Dammertz, H., Hanika, J., and Keller, A. 2008. Shallow bounding volume hierarchies for fast simd ray tracing of incoherent rays. *Computer Graphics Forum (Proceedings of EGSR 2008) 27,* 4.

[21] Dammertz, H., Sewtz, D., Hanika, J., and Lensch, H. P. A. 2010. Edge-avoiding a-trous wavelet transform for fast global illumination filtering. In *Proc. of the Conference on High Performance Graphics*. 67–75.

[22] Delbracio, M., Musé, P., Buades, A., Chauvier, J., Phelps, N., and Morel, J.-M. 2014. Boosting monte carlo rendering by ray histogram fusion. *ACM Trans. Graph. 33,* 1, 8:1–8:15.

[23] Dippé, M. A. Z. and Wold, E. H. 1985. Antialiasing through stochastic sampling. *SIGGRAPH Comput. Graph. 19,* 3, 69–78.

[24] Durand, F. and Dorsey, J. 2002. Fast bilateral filtering for the display of high-dynamic-range images. In *Proc. of the 29th annual conference on Computer graphics and interactive techniques*. SIGGRAPH '02. 257–266.

[25] Eisemann, E. and Durand, F. 2004. Flash photography enhancement via intrinsic relighting. *ACM Transactions on Graphics 23,* 3, 673–678.

[26] Eisemann, M., Bauszat, P., Guthe, S., and Magnor, M. 2012. Geometry presorting for implicit object space partitioning. *Computer Graphics Forum (Proc. of Eurographics Symposium on Rendering (EGSR)) 31,* 4 (June), 1445–1454.

[27] Eisemann, M., Bauszat, P., and Magnor, M. 2012. Implicit object space partitioning: The no-memory BVH. Tech. Rep. 16, Computer Graphics Lab, TU Braunschweig. Jan.

[28] Eisemann, M., Woizischke, C., and Magnor, M. 2008. Ray Tracing with the Single-Slab Hierarchy. In *Proceedings of Vision, Modeling, and Visualization (VMV'08)*. 373–381.

[29] Elad, M. 2002. On the origin of the bilateral filter and ways to improve it. *IEEE Transactions on Image Processing 11,* 10, 1141–1151.

[30] Ernst, M. and Greiner, G. 2007. Early split clipping for bounding volume hierarchies. In *Proc. of the IEEE Symposium on Interactive Ray Tracing*. 73–78.

[31] Ernst, M. and Greiner, G. 2008. Multi bounding volume hierarchies. In *Proc. of the IEEE Symposium on Interactive Ray Tracing*. 35 –40.

[32] Ernst, M. and Woop, S. 2011. Ray tracing with shared-plane bounding volume hierarchies. *Journal of Graphics, GPU, and Game Tools 15,* 3, 141–151.

[33] Fabianowski, B. and Dingliana, J. 2009. Compact BVH storage for ray tracing and photon mapping. In *Eurographics Ireland 2009*. 1–8.

[34] Farbman, Z., Fattal, R., Lischinski, D., and Szeliski, R. 2008. Edge-preserving decompositions for multi-scale tone and detail manipulation. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2008) 27,* 3.

[35] Fattal, R. 2009. Edge-avoiding wavelets and their applications. *ACM Trans. Graph. 28,* 3, 1–10.

[36] Feichtinger, H. G., Gröchenig, K., and Strohmer, T. 1995. Efficient numerical methods in non-uniform sampling theory. *Numer. Math. 69,* 4 (Feb.), 423–440.

[37] Garanzha, K., Pantaleoni, J., and McAllister, D. 2011. Simpler and faster hlbvh with work queues. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*. HPG '11. ACM, New York, NY, USA, 59–64.

[38] Gastal, E. S. L. and Oliveira, M. M. 2011. Domain transform for edge-aware image and video processing. *ACM TOG 30,* 4, 69:1–69:12. Proceedings of SIGGRAPH 2011.

[39] Gastal, E. S. L. and Oliveira, M. M. 2012. Adaptive manifolds for real-time high-dimensional filtering. *ACM TOG 31,* 4, 33:1–33:13. Proceedings of SIGGRAPH 2012.

[40] Günther, J., Popov, S., Seidel, H.-P., and Slusallek, P. 2007. Realtime ray tracing on GPU with BVH-based packet traversal. In *Proc. of the IEEE/Eurographics Symposium on Interactive Ray Tracing*. 113–118.

[41] Gunturk, B. K. 2010. Fast bilateral filter with arbitrary range and domain kernels. In *ICIP*. 3289–3292.

[42] Guo, J. and Pan, J. 2012. Point-wise Adaptive Filtering for Fast Monte Carlo Noise Reduction. In *Pacific Conference on Computer Graphics and Applications - Short Papers*. Hong Kong, 17–22.

[43] Hachisuka, T., Jarosz, W., Weistroffer, R. P., Dale, K., Humphreys, G., Zwicker, M., and Jensen, H. W. 2008. Multidimensional adaptive sampling and reconstruction for ray tracing. *ACM Trans. Graph. 27,* 3, 33:1–33:10.

[44] Havran, V., Herzog, R., and Seidel, H.-P. 2006. On fast construction of spatial hierarchies for ray tracing. Research Report MPI-I-2006-4-004, Max-Planck-Institut für Informatik. June.

[45] He, K., Sun, J., and Tang, X. 2010. Guided image filtering. In *Proc. of the European Conference on Computer Vision*. Vol. 1. 1–14.

[46] Hensley, J., Scheuermann, T., Coombe, G., Singh, M., and Lastra, A. 2005. Fast summed-area table generation and its applications. *Computer Graphics Forum 24*, 547–555.

[47] Ize, T. and Hansen, C. 2011. RTSAH traversal order for occlusion rays. *Computer Graphics Forum 30,* 2, 297–305.

[48] Kajiya, J. T. 1986. The rendering equation. In *Conference on Computer graphics and interactive techniques*. SIGGRAPH. 143–150.

[49] Kalantari, N. K. and Sen, P. 2011. Efficient computation of blue noise point sets through importance sampling. *Computer Graphics Forum 30,* 4, 1215–1221.

[50] Kalantari, N. K. and Sen, P. 2013. Removing the noise in monte carlo rendering with general image denoising algorithms. *Computer Graphics Forum 32,* 2pt1, 93–102.

[51] Karrenberg, R. 2007. *Memory Aware Realtime Ray Tracing: The Bounding Plane Hierarchy.* BA thesis, Universität des Saarlandes.

[52] Kay, T. L. and Kajiya, J. T. 1986. Ray tracing complex scenes. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques.* ACM Press, New York, NY, USA, 269–278.

[53] Keller, A. 1997a. Instant radiosity. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques.* SIGGRAPH '97. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 49–56.

[54] Keller, A. 1997b. Quasi-monte carlo methods for photorealistic image synthesis. Ph.D. thesis, Department of Computer Science, University of Kaiserslauten.

[55] Keller, A. and Wächter, C. 2009. Efficient ray tracing without acceleration data structure. *U.S. Patent Applications Publication No. US 2009/0225081 A1.*

[56] Kim, T.-J., Byun, Y., Kim, Y., Moon, B., Lee, S., and Yoon, S.-E. 2010. HCCMeshes: Hierarchical-culling oriented compact meshes. *Computer Graphics Forum (Eurographics) 29,* 2, 299–308.

[57] Kim, T.-J., Moon, B., Kim, D., and Yoon, S.-E. 2010. RACBVHs: Random-accessible compressed bounding volume hierarchies. *IEEE Transactions on Visualization and Computer Graphics 16,* 2, 273–286.

[58] Kipfer, P. and Westermann, R. 2005. Improved GPU sorting. In *GPUGems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*, M. Pharr, Ed. Addison-Wesley, 733–746.

[59] Kohavi, R. 1995. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence.* IJCAI'95. 1137–1143.

[60] Kosloff, T. and Barsky, B. 2007. An algorithm for rendering generalized depth of field effects based on simulated heat diffusion. In *Proc. ICCSA.* 1124–1140.

[61] Kraus, M. and Strengert, M. 2007. Depth-of-field rendering by pyramidal image processing. *Comput. Graph. Forum 26,* 3, 645–654.

[62] Lafortune, E. P. and Willems, Y. D. 1993. Bi-directional path tracing. In *Proc. of the third international conference on Computational Graphics and Visualization Techniques*. 145–153.

[63] Lauterbach, C., Garland, M., Sengupta, S., Luebke, D., and Manocha, D. 2009. Fast bvh construction on gpus. *Computer Graphics Forum 28,* 2, 375–384.

[64] Lauterbach, C., Yoon, S.-E., Tang, M., and Manocha, D. 2008. Reducem: Interactive and memory efficient ray tracing of large models. *Computer Graphics Forum 27,* 4, 1313–1321.

[65] Lee, S., Eisemann, E., and Seidel, H.-P. 2009. Depth-of-field rendering with multiview synthesis. *ACM Trans. Graph. 28,* 5 (Dec.), 134:1–134:6.

[66] Lee, S., Eisemann, E., and Seidel, H.-P. 2010. Real-time lens blur effects and focus control. *ACM Trans. Graph. 29,* 4 (July), 65:1–65:7.

[67] Lehtinen, J., Aila, T., Chen, J., Laine, S., and Durand, F. 2011. Temporal light field reconstruction for rendering distribution effects. *ACM Trans. Graph. 30,* 4.

[68] Lehtinen, J., Aila, T., Laine, S., and Durand, F. 2012. Reconstructing the indirect light field for global illumination. *ACM Trans. Graph. 31,* 4.

[69] Li, T.-M., Wu, Y.-T., and Chuang, Y.-Y. 2012. Sure-based optimization for adaptive sampling and reconstruction. *ACM Trans. Graph. 31,* 6 (Nov.), 194:1–194:9.

[70] MacDonald, D. J. and Booth, K. S. 1990. Heuristics for ray tracing using space subdivision. *Visual Computer 6,* 3, 153–166.

[71] Mahovsky, J. and Wyvill, B. 2006. Memory-conserving bounding volume hierarchies with coherent ray tracing. *Computer Grahics Forum 25,* 2 (June).

[72] McCool, M. D. 1999. Anisotropic diffusion for monte carlo noise reduction. *ACM Trans. Graph. 18,* 2, 171–194.

[73] Mehta, S., Wang, B., and Ramamoorthi, R. 2012. Axis-aligned filtering for interactive sampled soft shadows. *ACM Trans. Graph. 31,* 6, 163:1–163:10.

[74] Mehta, S. U., Wang, B., Ramamoorthi, R., and Durand, F. 2013. Axis-aligned filtering for interactive physically-based diffuse indirect lighting. *ACM Trans. Graph. 32,* 4 (July), 96:1–96:12.

[75] Mehta, S. U., Yao, J., Ramamoorthi, R., and Durand, F. 2014. Factored axis-aligned filtering for rendering multiple distribution effects. *ACM Trans. Graph. 33,* 4, 57:1–57:12.

[76] Mitchell, D. P. 1987. Generating antialiased images at low sampling densities. *SIGGRAPH Comput. Graph. 21,* 4, 65–72.

[77] Moon, B., Carr, N., and Yoon, S.-E. 2014. Adaptive rendering based on weighted local regression. *ACM Trans. Graph. 33,* 5 (Sept.), 170:1–170:14.

[78] Moon, B., Jun, J. Y., Lee, J., Kim, K., Hachisuka, T., and Yoon, S.-E. 2013. Robust image denoising using a virtual flash image for monte carlo ray tracing. *Computer Graphics Forum (Proc. of Eurographics Symposium on Rendering (EGSR)) 32,* 1, 139–151.

[79] Mora, B. 2011. Naive ray tracing: A divide-and-conquer approach. *ACM Transactions on Graphics*. Accepted for publication.

[80] Munkberg, J., Vaidyanathan, K., Hasselgren, J., Clarberg, P., and Akenine-Möller, T. 2014. Layered Reconstruction for Defocus and Motion Blur. *Computer Graphics Forum (Proceedings of EGSR 2014) 33,* 4, 81–92.

[81] Ooi, B., Sacks-David, R., and McDonnel, K. 1987. Spatial k-d-tree: An indexing mechanism for spatial databases. In *IEEE International Computer Software and Applications Conference (COMPSAC)*. Tokio, Japan, 433–438.

[82] Overbeck, R. S., Donner, C., and Ramamoorthi, R. 2009. Adaptive Wavelet Rendering. *ACM Transactions on Graphics (SIGGRAPH Asia 09) 28,* 5, 1–12.

[83] Pantaleoni, J. and Luebke, D. 2010. HLBVH: Hierarchical LBVH construction for real-time ray tracing of dynamic geometry. In *High Performance Graphics*. 87–95.

[84] Paris, S. and Durand, F. 2009. A fast approximation of the bilateral filter using a signal processing approach. *Int. J. Comput. Vision 81,* 1, 24–52.

[85] Park, H., Moon, B., Kim, S., and Yoon, S. 2013. P-RPF: pixel-based random parameter filtering for monte carlo rendering. In *2013 International Conference on Computer-Aided Design and Computer Graphics, CAD/Graphics 2013, Guangzhou, China, November 16-18, 2013*. 123–130.

[86] Parker, S., Martin, W., Sloan, P.-P. J., Shirley, P., Smits, B., and Hansen, C. 1999. Interactive ray tracing. In *Symposium on Interactive 3D Graphics*. 119–126.

[87] Parker, S. G., Bigler, J., Dietrich, A., Friedrich, H., Hoberock, J., Luebke, D., McAllister, D., McGuire, M., Morley, K., Robison, A., and Stich, M. 2010. Optix: A general purpose ray tracing engine. *ACM Transactions on Graphics*.

[88] Pérez, P., Gangnet, M., and Blake, A. 2003. Poisson image editing. *ACM Trans. Graph. 22,* 3, 313–318.

[89] Petschnigg, G., Szeliski, R., Agrawala, M., Cohen, M., Hoppe, H., and Toyama, K. 2004. Digital photography with flash and no-flash image pairs. *ACM Trans. Graph. 23,* 3, 664–672.

[90] Pham, T. Q. and Vliet, L. J. 2005. Separable bilateral filtering for fast video preprocessing. In *In IEEE Internat. Conf. on Multimedia and Expo*. 1–4.

[91] Pharr, M. and Humphreys, G. 2010. *Physically Based Rendering, Second Edition: From Theory To Implementation*, 2nd ed. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[92] Porikli, F. 2008. Constant time o(1) bilateral filtering. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on 0*, 1–8.

[93] Portilla, J., Strela, V., Wainwright, M. J., and Simoncelli, E. P. 2003. Image denoising using scale mixtures of gaussians in the wavelet domain. *IEEE Trans. Image Process 12*, 1338–1351.

[94] Ratanaworabhan, P., Ke, J., and Burtscher, M. 2006. Fast lossless compression of scientific floating-point data. In *Proc. of the Data Compression Conference*. 133–142.

[95] Rigau, J., Feixas, M., and Sbert, M. 2003. Refinement Criteria Based on f-Divergences. In *Eurographics Workshop on Rendering*. The Eurographics Association.

[96] Riguer, G., Tatarchuk, N., and Isidoro, J. R. 2003. Real-time depth of field simulation. In *ShaderX2: Shader Programming Tips and Tricks with DirectX 9.0*, W. Engel, Ed. Wordware.

[97] Rokita, P. 1996. Generating depth-of-field effects in virtual reality applications. *IEEE Comp. Graph. and its App. 16,* 2, 18–21.

[98] Rousselle, F., Knaus, C., and Zwicker, M. 2011. Adaptive sampling and reconstruction using greedy error minimization. *ACM Trans. Graph. 30,* 6, 159:1–159:12.

[99] Rousselle, F., Knaus, C., and Zwicker, M. 2012. Adaptive rendering with non-local means filtering. *ACM Trans. Graph. 31,* 6 (Nov.), 195:1–195:11.

[100] Rousselle, F., Manzi, M., and Zwicker, M. 2013. Robust denoising using feature and color information. *Comput. Graph. Forum 32,* 7, 121–130.

[101] Rubin, S. and Whitted, T. 1980. A 3-dimensional representation for fast rendering of complex scenes. In *7th annual conference on Computer graphics and interactive techniques*. SIGGRAPH. 110–116.

[102] Rudin, L. I., Osher, S., and Fatemi, E. 1992. Nonlinear total variation based noise removal algorithms. *Phys. D 60,* 1-4 (Nov.), 259–268.

[103] Samet, H. 2005. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann Publishers Inc.

[104] Segovia, B. and Ernst, M. 2010. Memory efficient ray tracing with hierarchical mesh quantization. In *Graphics Interface 2010*. 153–160.

[105] Sen, P. and Darabi, S. 2011. Compressive rendering: A rendering application of compressed sensing. *IEEE Transactions on Visualization and Computer Graphics 17,* 4, 487–499.

[106] Sen, P. and Darabi, S. 2012. On filtering the noise from the random parameters in monte carlo rendering. *ACM Trans. Graph. 31,* 3 (June), 18:1–18:15.

[107] Shirley, P., Aila, T., Cohen, J., E. E., Laine, S., Luebke, D., and McGuire, M. 2011. A local image reconstruction algorithm for stochastic rendering. In *Proceedings of the ACM Symposium on Interactive 3D Graphics and Games*.

[108] Soler, C., Subr, K., Durand, F., Holzschuch, N., and Sillion, F. 2009. Fourier depth of field. *ACM Trans. Graph. 28,* 2, 18:1–18:12.

[109] Stein, C. M. 1981. Estimation of the mean of a multivariate normal distribution. *The Annals of Statistics 9,* 6 (11), 1135–1151.

[110] Stich, M., Friedrich, H., and Dietrich, A. 2009. Spatial splits in bounding volume hierarchies. In *Proc. of High Performance Graphics*. 7–13.

[111] Szeliski, R. 2006. Image alignment and stitching: A tutorial. *Found. Trends. Comput. Graph. Vis. 2,* 1, 1–104.

[112] Tomasi, C. and Manduchi, R. 1998. Bilateral filtering for gray and color images. In *Proc. of the International Conference on Computer Vision*. 839–846.

[113] Vaidyanathan, K., Munkberg, J., Clarberg, P., and Salvi, M. 2014. Layered light field reconstruction for defocus blur. *To appear in ACM Transactions on Graphics*.

[114] Veach, E. and Guibas, L. J. 1997. Metropolis light transport. In *Computer Graphics*. SIGGRAPH '97. 65–76.

[115] Wächter, C. 2008. Quasi-monte carlo light transport simulation by efficient ray tracing. Ph.D. thesis, Universität Ulm.

[116] Wächter, C. and Keller, A. 2006. Instant ray tracing: The bounding interval hierarchy. In *Eurographics Symposium on Rendering*. 139–149.

[117] Wächter, C. and Keller, A. 2006. Instant Ray Tracing: The Bounding Interval Hierarchy. In *Rendering Techniques 2006 (Eurographics Symposium on Rendering)*. 139–149.

[118] Wächter, C. and Keller, A. 2007. Terminating spatial hierarchies by a priori bounding memory. In *Proc. of IEEE Symposium on Interactive Ray Tracing*. 41 –46.

[119] Wald, I. 2004. Realtime Ray Tracing and Interactive Global Illumination. Ph.D. thesis, Computer Graphics Group, Saarland University.

[120] Wald, I. 2007. On fast Construction of SAH based Bounding Volume Hierarchies. *Eurographics/IEEE Symposium on Interactive Ray Tracing*, 33–40.

[121] Wald, I., Benthin, C., Wagner, M., and Slusallek, P. 2001. Interactive rendering with coherent ray tracing. *Computer Graphics Forum 20,* 3, 153–164.

[122] Wald, I., Boulos, S., and Shirley, P. 2007. Ray Tracing Deformable Scenes using Dynamic Bounding Volume Hierarchies. *ACM Transactions on Graphics 26,* 1, 1–28.

[123] Wald, I., Dietrich, A., and Slusallek, P. 2005. An interactive out-of-core rendering framework for visualizing massively complex models. In *ACM SIGGRAPH 2005 Courses*.

[124] Wald, I., Kollig, T., Benthin, C., Keller, A., and Slusallek, P. 2002. Interactive global illumination using fast ray tracing. In *Proc. of the 13th Eurographics Workshop on Rendering*. 15–24.

[125] Wald, I., Mark, W. R., Günther, J., Boulos, S., Ize, T., Hunt, W., Parker, S. G., and Shirley, P. 2009. State of the Art in Ray Tracing Animated Scenes. *Computer Graphics Forum 28,* 6, 1691–1722.

[126] Wang, Z. and Bovik, A. 2009. Mean squared error: Love it or leave it? A new look at Signal Fidelity Measures. *IEEE Signal Processing Magazine 26,* 1, 98 –117.

[127] Wang, Z., Bovik, A. C., Sheikh, H. R., and Simoncelli, E. P. 2003. The ssim index for image quality assessment. *MATLAB implementation available online from: http://www. cns. nyu. edu/lcv/ssim 23*, 66.

[128] Woop, S., Feng, L., Wald, I., and Benthin, C. 2013. Embree ray tracing kernels for cpus and the xeon phi architecture. In *ACM SIGGRAPH 2013 Talks*. SIGGRAPH '13. ACM, New York, NY, USA, 44:1–44:1.

[129] Woop, S., Marmitt, G., and Slusallek, P. 2006. B-KD Trees for Hardware Accelerated Ray Tracing of Dynamic Scenes. In *Proceedings of Graphics Hardware*. 67–77.

[130] Xu, R. and Pattanaik, S. N. 2005. A novel monte carlo noise reduction operator. *IEEE Comput. Graph. Appl. 25,* 2, 31–35.

[131] Yoon, S.-E., Lauterbach, C., and Manocha, D. 2006. R-LODs: fast LOD-based ray tracing of massive models. *The Visual Computer 22,* 9-11, 772–784.

[132] Zachmann, G. 2002. Minimal hierarchical collision detection. In *VRST '02: Proceedings of the ACM symposium on Virtual reality software and technology*. ACM, New York, NY, USA, 121–128.

[133] Zhou, T., Chen, J. X., and Pullen, M. 2007. Accurate depth of field simulation in real time. *Comput. Graph. Forum 26,* 1, 15–23.

[134] Zuniga, M. and Uhlmann, J. 2006. Ray queries with wide object isolation and the de-tree. *Journal of Graphics Tools 11,* 3, 27–45.

Curriculum Vitæ - Lebenslauf

Curriculum Vitæ

1983	born in Gießen, Germany
1989 - 1993	Elementary school
	Erich-Kästner Grundschule, Wolfsburg, Germany
1994 - 2002	Abitur, main subjects physics and mathematics
	Heinrich-Nordhoff Gesamtschule, Wolfsburg, Germany
2002 - 2003	Civil Service
	Sports Club TV Jahn Wolfsburg, Wolfsburg, Germany
2003 - 2011	Diploma in Computer Science
	TU Braunschweig, Germany
2011 - 2015	Ph.D. Student Computer Science, Prof. M. Magnor
	TU Braunschweig, Germany

Lebenslauf

1983	geboren in Gießen, Deutschland
1989 - 1993	Grundschule
	Erich-Kästner Grundschule, Wolfsburg, Deutschland
1994 - 2002	Abitur, Leistungskurse Physik und Mathematik
	Heinrich-Nordhoff Gesamtschule, Wolfsburg, Deutschland
2002 - 2003	Zivildienst
	Sportverein TV Jahn Wolfsburg, Wolfsburg, Deutschland
2003 - 2011	Diplom im Studiengang Informatik
	TU Braunschweig, Deutschland
2011 - 2015	Ph.D. Student Informatik, Prof. M. Magnor
	TU Braunschweig, Deutschland

Publications

Journal Articles

- Michael Stengel, Pablo Bauszat, Martin Eisemann, Elmar Eisemann, and Marcus Magnor.
  Temporal Video Filtering and Exposure Control for Perceptual Motion Blur.
  In *Transactions on Visualization and Computer Graphics (TVCG)*, 2014.

- Pablo Bauszat, Martin Eisemann, Stefan John, and Marcus Magnor.
  Sample-Based Manifold Filtering for Interactive Global Illumination and Depth of Field.
  In *Computer Graphics Forum*, 34(1):265–276, 2014.

- Oskar Elek, Pablo Bauszat, Tobias Ritschel, Marcus Magnor, and Hans-Peter Seidel.
  Spectral Ray Differentials.
  In *Computer Graphics Forum (Proc. of Eurographics Symposium on Rendering (EGSR))*, 33(4), 2014.
  Received the Best Student Paper award at EGSR 2014

- M. Eisemann, P. Bauszat, S. Guthe, M. Magnor.
  Geometry Presorting for Implicit Object Space Partitioning.
  In *Computer Graphics Forum (Proc. of Eurographics Symposium on Rendering (EGSR))*, 31(4):1445–1454, 2012.

- P. Bauszat, M. Eisemann, M. Magnor.
  Guided Image Filtering for Interactive High-quality Global Illumination.

In *Computer Graphics Forum (Proc. of Eurographics Symposium on Rendering (EGSR))*, 30(4):1361–1368, 2011.

International, Refereed Conferences

- Lorenz Rogge, Pablo Bauszat, and Marcus Magnor.
  Monocular Albedo Reconstruction.
  In *International Conference on Image Processing (ICIP)*, 2014.

- Oskar Elek, Pablo Bauszat, Tobias Ritschel, Marcus Magnor, and Hans-Peter Seidel.
  Progressive Spectral Ray Differentials.
  In *Vision, Modeling and Visualization (VMV)*, pages 151–158, 2014.

- P. Bauszat, M. Eisemann, M. Magnor.
  Adaptive Sampling for Geometry-aware Reconstruction Filters.
  In *Vision, Modeling and Visualization (VMV)*, pages 183–190, 2011.

- P. Bauszat, M. Eisemann, and M. Magnor.
  The Minimal Bounding Volume Hierarchy.
  In *Vision, Modeling and Visualization (VMV)*, pages 227–234, 2010.

Technical Reports

- M. Eisemann, P. Bauszat, and M. Magnor.
  Implicit Object Space Partitioning: The No-Memory BVH.
  Technical Report 2012-01-16, Computer Graphics Lab, TU Braunschweig, 2012.

Refereed Posters

- Pablo Bauszat, Marc A. Kastner, Martin Eisemann, and Marcus Magnor.
  The Split Grid - A Hierarchical 1D-Grid-based Acceleration Data Structure for Ray Tracing.
  Poster at Eurographics 2014, March 2014.